TRABAJO FIN DE GRADO

Grado en ingeniería en tecnologías industriales

Enchufe domótico
Smartplug : El enchufe inteligente

Autora : Susana Niclós Ferreras

Tutor : Pablo Zumel Vaquero

Fecha de Presentación: Septiembre 2014

Índice general

Agradecimientos

En este apartado me gustaría dar las gracias no solo a las personas que han contribuido en este proyecto sino a todas las que me han ayudado a llegar hasta aquí.

Primero me gustaría dar las gracias a mi padre Vicente que con sus tortugas inspiró este proyecto, y que con su constante apoyo hizo posible llevarlo a cabo. Además me gustaría agradecer a mi madre Margarita y a mi hermana Ester, que no solo han colaborado en la revisión de esta memoria sino que siempre han estado presentes en todos los aspectos de mi vida, dispuestas a ayudar y apoyarme.

Seguidamente me gustaría agradecer a mi cuñado Miguel Ángel por contribuir con sus conocimientos a mejorar este proyecto, a mi vecina Irene por colaborar con la fotografía y edición de vídeo y a mi novio Javier por corregir esta memoria, y por aportar inspiración e ideas a este proyecto.

Como no, agradecer a mi abuelo Román quien desde pequeña me enseñó el arte de las matemáticas, la física y el dibujo, que dedicó cientos de horas a mi formación y sin el cual nunca habría llegado a ser ingeniero, y a mi abuela Pilar siempre dispuesta a animarnos y alimentarnos en esas largas horas de clases.

Finalmente me gustaría darle las gracias a mi tutor Pablo Zumel por asistirme durante todo el proceso, y ayudarme con los problemas que han ido surgiendo.

Resumen

El Smartplug es un sistema capaz de controlar de forma remota cuatro enchufes a través de un entorno web. Está concebido para ofrecerse como un producto finalizado o para el desarrollo de nuevas aplicaciones basadas en él.

El módulo estándar (Smartplug) integra un servidor web que permite programar remotamente los enchufes de forma independiente de las siguientes maneras: programación de los enchufes mediante intervalos horarios, encendido y apagado manual. Además es fácilmente combinable con módulos específicos, estos contienen los distintos sensores y le proporcionan funcionalidades adicionales.

La electrónica que compone el módulo estándar es: el Arduino Yún, una placa de relés, una fuente de alimentación y un reloj en tiempo real. En cuanto al software, esta desarrollado utilizando los lenguajes: C++, Python, HTML5, CSS3, y Javascript.

El Smartplug es sencillo de utilizar y no requiere instalación, solo hay que enchufarlo y para utilizar la aplicación que sirve para controlarlo solo es necesario tener un dispositivo con un navegador que soporte HTML5.

Se ha dedicado especial esfuerzo a que la aplicación para el control remoto funcione independientemente del dispositivo y del sistema operativo. Se puede utilizar en ordenadores, tablets y smartphones sin necesidad de agregar en ellos ningún software.

Es importante tener en cuenta que el Smartplug es un proyecto abierto, en el que el usuario tiene acceso a todo el código fuente. Para facilitar el desarrollo de nuevas aplicaciones se han creado dos módulos especializados totalmente operativos (el módulo control ambiental y el módulo de alarma) especialmente documentados para servir de guía.

Antes de comenzar el desarrollo del Smartplug se hizo un estudio de mercado para evaluar las características de los dispositivos similares, de estos se extrajeron las siguiente conclusiones: los precios oscilan entre 50-60 €, solo tienen un enchufe, y por regla general carecen de sensorización. Además a diferencia del Smartplug son productos cerrados, que no permiten ni desarrollo ni personalización.

Actualmente hay tres prototipos totalmente funcionales y probados, un módulo estándar y dos módulos especializados : control ambiental (Smartplug01) y alarmas (Smartplug02).

Abstract

Smartplug is a wireless device capable of controling four plugs remotely using a web environment. It will be available ready to use and a as developer kit to create new applications based on it.

The standard module (Smartplug) integrates a web server that enables the independent and remote programing of the plugs in different ways: Fixed time intervals, manual on, and manual off. In addition, it is easily to combine with specialized modules that contain different sensors and add new functionalities.

The electronic devices included in the standard module are: Arduino Yún board, a 220VAC to 5V transformer, a relay board, and a real time clock. The base software has been developed using the following programming languages : C++, Python, HTML5, CSS3 and Javascript.

Smartplug is easy to use, and installation it's not required, you just simply have to plug it into a power socket. Also, the remote controlling app neither needs to be installed.

Great care was taken to ensure that the app works on any device with a HTML5 compatible browser. The app can be used independently of the type of operating system or the screen size. It runs on computers, tablets and smartphones.

Its important to notice that this is an open proyect and all the code can be downloaded from the web. In fact in order to help the development of new applications, two specialized modules have been created (environment control and alarm system) and carefully been documented as a tutorial system.

Before we started the development of Smartplug, a market survey was made to evaluate devices with similar characteristics, from that the following conclusions were drawn: Prices range from 50-60 €, they only have one plug, and usually the lack from sensors. Besides, most of them are closed products and don't allow any development or customization.

Currently we have been built three totally working prototipes, one standard module and two specialized modules: Smartplug01 an enviroment control system, and Smartplug02 an alarm system.

Capítulo 1

Introducción

1.1. Breve descripción

En este trabajo de fin de grado se ha desarrollado un sistema capaz de controlar cuatro enchufes programables de forma remota a través de la web, y sobre el que se pueden desarrollar infinidad de aplicaciones añadiendo módulos especializados. Este sistema se ha denominado Smartplug.

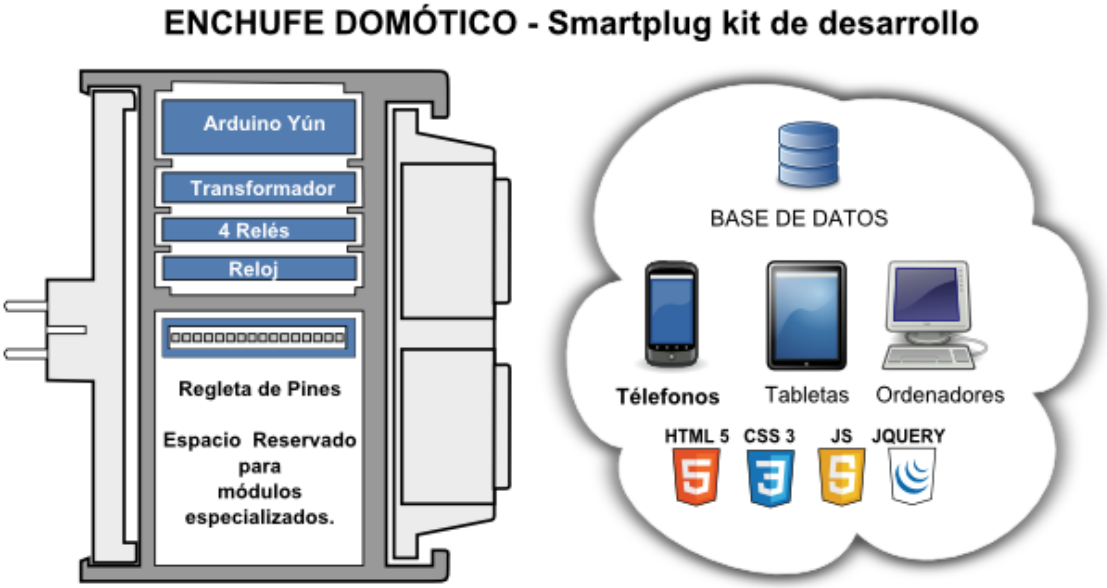


Figura 1.1: Esquema lógico del Smartplug

Tiene las siguientes características:

Ampliable: Su desarrollo modular hace que sea fácilmente integrable con módulos especializados que le otorgan nuevas funcionalidades.

Preparado para desarrolladores: El Smartplug o módulo estándar es una placa con las funcionalidades básicas, especialmente elaborada para que los desarrolladores puedan crear sus propias aplicaciones.

Se han desarrollado dos aplicaciones o módulos específicos (Control ambiental, y gestión de alarmas) con el fin de servir de guía de desarrollo.

Libre: Es un entorno abierto, todo el código fuente es libre y se encuentra comentado para facilitar el desarrollo de nuevos proyectos.

Conectar y listo: Solo es necesario enchufarlo, no necesita ningún tipo de infraestructura física. Además, tampoco requiere la instalación de software, la gestión remota tan solo requiere de un navegador web que soporte HTML5. Toda su gestión y control se realiza mediante un servidor web que el mismo integra.

Accesibilidad: Las páginas están desarrolladas con estilo adaptable a diferentes tipos de pantalla, se puede utilizar un smartphone, tablet o pc, con sistemas operativos como: Linux, Windows, Android, Mac OS...

Conectividad: Puede actuar como punto de acceso y como cliente WIFI, lo que le permite un acceso desde Internet.

1.2. Motivación

Este proyecto nació a principios de 2013 con el objetivo de controlar un terrario de forma remota y económica combinando diferentes dispositivos electrónicos de uso general. La idea era crear un producto capaz de conectar/desconectar automáticamente los dispositivos que tenía enchufados (calefactor, humidificador y ventilador) según los valores de temperatura y humedad que presentara el terrario y controlable con una aplicación WIFI.

Sin embargo, rápidamente se vio que este proyecto tenía muchas más posibilidades si se desarrollaba como un módulo generalista o estándar combinable con otros módulos que le especializaran. Así, la idea original solo sería una de las múltiples aplicaciones a las que se podía adaptar el producto, y simplemente cambiando el módulo especializado cambiaría totalmente de funcionalidades.

Actualmente se ha desarrollado el módulo estándar o Smartplug, y se ha combinado con éxito con dos módulos especializados: El módulo de control Ambiental (Smartplug01) y el módulo de sistemas de alarma (Smartplug02)

Por ahora este proyecto aún se encuentra en una primera fase de desarrollo, su precio y gran tamaño no lo hacen adecuado para la comercialización, sin embargo se prevé en la segunda fase conseguir mejorar ampliamente estos factores.

1.3. Estructura del documento

Este documento se ha dividido en nueve capítulos que se describen a continuación:

En el primer capítulo se ha escrito una breve introducción sobre qué es el Smartplug y como surgió esta idea.

En el segundo capítulo se describen los conceptos básicos de la domótica, se hace un estudio del estado del arte, y tras evaluar el mercado actual se definen los requisitos y restricciones del problema y se plantean soluciones.

El tercer capítulo se encarga de definir el hardware que se va a utilizar en el módulo estándar y sus especificaciones relevantes.

En el cuarto capítulo se hace una descripción detallada de como esta compuesto el módulo estándar, se describen las conexiones, y el software desarrollado.

El quinto y sexto capítulo sirven de guía para desarrollar nuevas aplicaciones basadas en el módulo estándar, en ellos se describe el proceso que se ha seguido para desarrollar los dos módulos especializados, las conexiones y modificaciones en la programación.

El séptimo capítulo contiene las dos planificaciones que se realizaron, una primera con estimaciones, y una final con los datos reales. Estas planificaciones se han utilizado para en el octavo capítulo calcular el presupuesto estimado y el real.

En el noveno capítulo se encuentran las conclusiones obtenidas tras terminar este proyecto, y las posibles mejoras que se podrían aplicar en una segunda fase.

Finalmente en la última parte de la memoria se encuentran los apéndices, que consisten en unas breves guías sobre los distintos software empleados, los planos de las piezas de la carcasa, y los diagramas de Gantt ampliados. A lo largo de la memoria en distintos apartados se hace referencia a estos apéndices para ampliar la información.

Capítulo 2

Planteamiento del problema: análisis del estado del arte, requisitos y restricciones

2.1. Introducción a la domótica

Actualmente la domótica está tomando mucha importancia en el mundo, con el avance de la tecnología es posible realizar muchas tareas de forma automática, mejorando así la calidad de vida de las personas y la eficiencia de los procesos.

Para entender bien este proyecto se debe empezar por definir que significa exactamente este término. Se entiende por domótica el conjunto de sistemas que automatizan las diferentes instalaciones de una vivienda [1].

Estructura de un sistema domótico

El concepto de domótica es muy amplio, recoge desde tener un único dispositivo encargado de una tarea, hasta sistemas capaces de controlar una vivienda [2].

Sin embargo casi todos los sistemas domóticos siguen esta estructura:

1. **Controladores:** Encargados de gestionar el sistema de acuerdo a la programación que contienen y según la información leída de los sensores.
2. **Sensores:** Capturan datos del entorno y los transfieren al controlador.
3. **Actuadores:** Reciben la orden enviada desde el controlador y la llevan a cabo.
4. **Bus:** Encargados de la transmisión de información, aunque también se puede realizar de forma inalámbrica.
5. **Interfaz:** Dispositivos (PC, móvil, tablet) que interactúan con el sistema domótico.

Arquitectura

Según el modo en que se haya diseñado el circuito de control existen dos arquitecturas básicas [4]:

1. **Centralizada:** Posee un único controlador central que recibe la información de todos los sensores, la procesa según su programación y envía las ordenes a los actuadores.
2. **Distribuida:** Cada grupo de sensor y actuador posee un controlador propio capaz de leer, procesar y ejecutar las ordenes según la programación.

Nota: También existen arquitecturas mixtas entre estos dos sistemas.

Modo de transmisión

Según el modo en que se transfiera la información entre las distintas partes del sistema, existen diferentes tipos de sistemas domóticos [4]:

1. **Cableado Propio:** El sistema cuenta con su propio circuito, requiere mucha instalación pero es el más común .
2. **Cableado Compartido:** Utiliza las instalaciones existentes en la vivienda para transmitir su información, como el cable de teléfono o el de red.
3. **Inalámbrica:** No requiere cables. Actualmente está cobrando mayor importancia gracias a incremento de uso de las redes 3G, y 4G que permiten a los usuarios desde cualquier lugar acceder a su dispositivo domótico.

Niveles de domotización

Además, existen diferentes niveles de domotización [3]:

1. **Hogares con objetos automatizados:** Contienen objetos que realizan de forma automática labores, por ejemplo una lavadora.
2. **Hogares con objetos automatizados y comunicación:** Contienen objetos capaces de realizar de forma automática labores y de comunicarse entre ellos para mejorar su funcionalidad. Por ejemplo automatizar el encendido de luces utilizando sensores de presencia.
3. **Hogares conectados**: Poseen redes externas que les permiten controlar remotamente los objetos e interactuar con ellos para obtener información y servicios. Por ejemplo el Smartplug.
4. **Hogares con capacidad de aprender:** Guardan registros constantemente para buscar patrones y ser capaces de anticiparse a las necesidades de los usuarios. Un ejemplo sería las neveras que avisan al usuario al detectar falta de los productos que este más utiliza.

Protocolos de domótica

El protocolo de comunicación es la forma en que se lleva a cabo el intercambio de datos entre los dispositivos de un sistema domótico [5].

Actualmente existen una gran variedad de ellos, algunos especializados en la domótica y otros adaptados a ella. Estos protocolos se pueden agrupar en dos categorías:

1. **Protocolos Propietarios:** Han sido creados por una empresa y son para uso exclusivo del fabricante [5].
2. **Protocolos Estándar:** Se utilizan en varias empresas que fabrican dispositivos compatibles entre sí. Pueden ser con licencia o de uso libre [5].

Algunos protocolos importantes son:

1. **X-10:** Desarrollado a finales de los años 70, es uno de los protocolos más antiguos. Su función es el control de electrodomésticos utilizando la red eléctrica para transmitir las ordenes. Es uno de los más sencillos ya que no requiere instalación, y puede resultar muy útil para aparatos que sean compatibles [6].
2. **EHS:** Ha sido creado por la comisión Europea con el fin de aumentar la implementación de la tecnología domótica en el mercado residencial Europeo. Es un protocolo abierto basado en OSI (Open Standard Interconnectio) en el que especifican los niveles : físico, de enlace de datos, de red, y de aplicación [7].
3. **EIB:** También de desarrollo Europeo, esta basado en un bus de datos, y requiere su propio cableado [7].
4. **BIODOM:** Este protocolo ha sido creado por españoles basándose en el estándar EHS y se puede adaptar con facilidad a nuevas necesidades [8].
5. **Lonworks:** Es un protocolo propio de la empresa americana Echelon Corporation. Puede utilizar distintos medios de trasmisión, requiere mucha instalación pero es una tecnología muy resistente y fiable [9].

Empresas y proyectos domóticos

Existen infinidad de compañías dedicadas a la domótica tanto a nivel mundial como nacional, algunos ejemplos son: mControl, HomeSeer, Domótica Soluciones Integrales (Española)...

Sin embargo son GOOGLE y Apple quienes están llevando a cabo las mayores labores de investigación en este campo, aunque sus proyectos aún no están a la venta.

1. **Proyecto GOOGLE:** En el año 2011 GOOGLE anuncio Android@home, un sistema domótico controlado a través de dispositivos Android, sin embargo actualmente no existe mucha información del proyecto [10].

2. **Proyecto Apple:** En Junio de 2014 Apple anunció HomeKit, una aplicación para hogares inteligentes. Esta aplicación será compatible con la mayor parte de los electrodomésticos y dispositivos inteligentes, y será capaz de controlarlos de distintas formas [11].

2.2. Aplicaciones para smartphones, tablets y ordenadores

Actualmente las ventas de smartphones y tablets se han incrementado exponencialmente. Dispositivos con iOS y Android ocupan el 75 % del mercado y la mayor parte de ellos cuentan con una conexión móvil de Internet

Estos dispositivos tienen acceso a infinidad de aplicaciones, que les permiten desde cualquier lugar realizar una gran variedad de tareas desde escuchar música, hasta reservar un viaje o consular el tiempo.

A la hora de diseñar una aplicación existen distintas opciones de diseño:

1. **Compatibles con Android**: Aplicaciones diseñadas específicamente para el sistema operativo de GOOGLE llamado Android del que existen varias versiones. Para ofrecerlas al público es necesario la aprobación de GOOGLE.

2. **Compatibles con iOS**: Aplicaciones diseñadas específicamente para dispositivos de marca Apple (iPhone y iPad), al igual que la anterior para ofrecerlas al público es necesario la aprobación de Apple.

3. **Compatibles con Windows Phone**: Para dispositivos con ese sistema operativo.

4. **Compatibles con cualquier dispositivo**: Aplicaciones en HTML5 que funcionan desde cualquier sistema operativo siempre y cuando este posea un navegador compatible. Estas aplicaciones son visualizables desde PC, tablets y móviles.

2.3. Estado del arte

Actualmente existen una gran cantidad de dispositivos similares al Smartplug, capaces de controlar enchufes utilizando una aplicación móvil.

En el ultimo año han aparecido modelos de la marcas D-Link y Belkin, y grandes empresas como Samsung o Siemens están desarrollando proyectos semejantes.

Por ello antes de establecer los requisitos de este proyecto es necesario estudiar las características de estos dispositivos para poder desarrollar un producto mejor.

A continuación se van a comparar algunos de los enchufes inteligentes más importantes del mercado actualmente:

D-Link DSP-W215

Este dispositivo esta compuesto por solo un 1 enchufe y tiene un precio de 50 €. Se controla a través de la WIFI con una aplicación Android que permite encender o apagar el dispositivo o programarlo con intervalos horarios. No requiere ningún soporte adicional, es capaz de hacer estadísticas del uso de energía y contiene un sensor de temperatura que en caso de sobrecalentamiento apaga automáticamente el aparato [12].

Figura 2.1: Enchufe D-Link [12]

Belkin WEMO

Este enchufe inteligente tiene un precio de 60€, y permite mediante una aplicación programar horarios u otras normas de encendido, también es capaz de crear estadísticas del gasto energético. Este enchufe ofrece la posibilidad de ser combinado con otros productos de la misma marca (bombillas, sensores de presencia...) para aumentar las posibilidades de programación, aunque esto conlleva un gasto adicional [13].

Figura 2.2: Enchufe Belkin [13]

EasySmart

Este dispositivo quizás no es de los más importantes, pero se ha considerado digno de mención ya que es de los pocos que contienen más de un enchufe, en concreto tiene 6 y un precio de 45 €. Posee una aplicación para conectar o desconectar los dispositivos utilizado la red WIFI, pero no es posible programarlo [14].

Nota: Aunque existen más dispositivos similares al Smartplug, estos presentan características muy parecidas a los descritos

Figura 2.3: Enchufe Easysmart [14]

anteriormente por lo que no se han añadido.

Resumen de Características

Producto	Precio	Nº Enchufes	Programación	Opciones
D-Link DSP-W215	50	1	Horaria	Estadísticas, y protección frente sobrecalentamiento.
Belkin WEMO	60	1	Horaria y normas	Estadísticas y combinable con módulos .
EasySmart	45	6	No tiene	

Tabla 2.1: Comparativa de los enchufes inteligentes más significativos del mercado

2.4. Análisis del problema: requisitos y restricciones

Tras haber estudiado los productos del mercado que resultan relevantes, es necesario combinar las conclusiones con la idea original para definir más los requisitos y restricciones del proyecto.

El objetivo original de este proyecto era desarrollar un dispositivo inteligente capaz de controlar enchufes según la información recibida por un reloj en tiempo real y posibles sensores adicionales, y que permitiera con una aplicación realizar y supervisar la programación a través de la red WIFI.

Al igual que los dispositivos existentes debe ser sencillo de utilizar y permitir realizar todas las programaciones remotamente, es decir, a través de la red WIFI.

El problema con la domótica es que en la mayoría de las ocasiones para realizar la automatización del hogar es necesario reformar la casa, se deben cambiar la mayoría de los dispositivos para incorporar tecnología inteligente en ellos. Sin embargo, en este proyecto se requiere que el Smartplug sea válido para cualquier hogar sin necesidad de cambios, ni instalaciones.

Se requiere que sea compatible con cualquier tipo de aparato que se enchufe a la corriente, y no solo aquellos que utilicen ciertos protocolos, ya que esto permite que sea válido para cualquier hogar.

Además se quiere que sea modular como el de Belkin pero con un concepto mucho más abierto, que sea combinable con placas de sensores del fabricante pero también con placas diseñadas por el usuario. Así se busca conseguir un producto adecuado tanto para desarrolladores como para usuarios que lo quieran listo para funcionar.

Con el objetivo de ofrecer mejores características que los enchufes existentes se ha optado por tener 4 enchufes con programaciones independientes y ampliables según los módulos personalizados que contenga.

Además se busca que sea económico, seguro y que pueda controlarse desde cualquier dispositivo con un navegador de Internet y conexión WIFI, independientemente del sistema operativo. Se requiere que sea visualizable correctamente desde ordenadores, tablets y smartphones, es decir, que la aplicación sea adaptable a cualquier tamaño de pantalla.

2.5. Normativas legales a cumplir

Como se indica en la guía técnica del Reglamento Electrotécnico de Baja Tensión (REBT) en el apartado de instalaciones domóticas, el Smartplug debe cumplir con la normativa UNE-EN 50090-2-2.

Esta normativa garantiza que el dispositivo funciona sin provocar perturbaciones electromagnéticas al entorno. El dispositivo debe producir perturbaciones por debajo del nivel estipulado, y ser capaz de soportar perturbaciones de hasta este nivel.

Además el Smartplug debe cumplir con el estándar internacional IEEE 802.11, publicado por el Instituto de Ingenieros Eléctricos y Electrónicos. Esta normativa técnica regula los sistemas WIFI, en ella se especifican las normas de funcionamiento y características que deben seguir las redes y dispositivos de este tipo.

La mayoría de los dispositivos WIFI comerciales siguen IEEE 802.11a o IEEE 802.11b. La primera utiliza la banda de 5 GHz, y estimula una velocidad máxima de transmisión 54Mbps. La segunda al utilizar la banda de 2,4GHz que también es utilizada por otros dispositivos (monitores de bebe, microondas...) esta limitada a una velocidad máxima de 11Mbps.

En este proyecto no van a existir problemas para cumplir con ninguna de esas normativas, ya que se van a utilizar dispositivos comerciales ya certificados.

2.6. Soluciones propuestas

Una vez analizados los requisitos comienza la fase de diseño. Como cualquier sistema domótico este dispositivo estará compuesto de un controlador, actuadores, sensores, un medio de transmisión y un interfaz. Además también se requiere una carcasa para hacerlo seguro y resistente.

Es importante tener en cuenta a la hora de diseñar que uno de los requisitos más importantes es el desarrollo modular, el objetivo principal de este proyecto es desarrollar un módulo estándar generalista sobre el que poder añadir módulos especializados con sensores y funciones adicionales.

2.6.1. Elección del controlador

Para la realización de este proyecto es necesario utilizar un microcontrolador capaz de procesar la información obtenida por los sensores, y ejecutar las peticiones enviadas por el usuario a través del entorno web. Así se requiere tener una placa con conexión WIFI, que pueda actuar como servidor y con una disponibilidad de alrededor de 15 puertos de propósito general y al menos 4 analógicos.

Lo ideal sería diseñar una placa que se adaptara exactamente a las necesidades del proyecto lo que permitiría reducir costes y tamaño, sin embargo para el primer prototipo se ha optado por utilizar una placa comercial, por lo que una vez comprendidos los requerimientos se ha realizado un estudio entre las distintas placas del mercado.

Requerimientos	Arduino Yún [15]		Raspberry pi B [16]	BeagleBone [17]
Tamaño (cm)	7.5 x 5.3		8.6 x 5.7	8.6 x 5.3
Procesador	At6mega32U4	Atheros AR9331	ARM11	ARM Cortex-A8
Frec. reloj	16MHz	MIPS@400MHz	700 Mhz	700MHz
Ram	2.5 KB	64 MB	512 MB	256 MB
Flash	32 KB	16 MB y microSD	microSD	4GB y microSD
Puertos GPIO	20		8	66
P. analógicos	12 10-bit		N/A	7 12-bit
Conexiones	WIFI y Ethernet		Ethernet	Ethernet
Precio (€)	65		35	70

Tabla 2.2: Comparativa de placas controladoras disponibles en el mercado

Tras comparar las características individuales de cada una de las placas se ha optado por utilizar el Arduino Yún, ya que se adapta a la perfección a los requerimientos técnicos al tener disponible una conexión WIFI sin necesidad de añadir otro dispositivo, poder actuar como servidor y al tener una suficiente cantidad de puertos GPIO y analógicos. Además, al contrario que en las otras placas, estos puertos en Arduino Yún cuentan con protecciones contra sobreintensidades.

Finalmente Arduino tiene disponible una gran cantidad de documentación existente lo que facilita el desarrollo del proyecto.

2.6.2. Elección de actuadores

Los actuadores en este proyecto deben ser capaces de recibir las ordenes enviadas desde los puertos digitales del microcontrolador y ejecutarlas para conectar o desconectar enchufes.

Para realizar esto, desde un primer momento se decidió utilizar relés, ya que al aislar los circuitos proporcionan mayor seguridad. Para el montaje primero se estudió la posibilidad de comprar los relés y elaborar el circuito necesario, sin embargo dado el bajo coste de las placas con los circuitos ya preparados y listas

para conectar y dado la comodidad que proporciona una placa ya probada se opto por esta opción.

En el mercado existen numerosas placas de relés compatibles con Arduino, tras estudiarlas brevemente y concluir que las diferencias en precio y especificaciones eran muy pequeñas se opto por utilizar la Ywrrobot 4 relay board, aunque cualquier placa similar sería perfectamente válida.

2.6.3. Elección del modo de integración de los sensores

En cuanto a los sensores, las decisiones de diseño no son sobre del tipo de sensores sino sobre el modo en que estos se integran con el módulo base.

Cuando se comenzó el desarrollo se pensó la opción de elaborar una serie de Smartplug independientes cada uno con sus sensores y funciones distintas, pero rápidamente se decidió que tenía mucha más posibilidades darle al usuario final un módulo general que poder personalizar.

El módulo general es el núcleo del Smartplug contiene toda la electrónica común, sobre él se construyen todos los módulos especializados con sus sensores y elementos adicionales.

Además en la segunda fase de desarrollo se realizará una placa que contenga el módulo estándar con un lugar para conectar las placas especializadas. Como se trata de un proyecto abierto el usuario podrá diseñarse su propia placa o comprar un prototipo ya montado y personalizado.

2.6.4. Interfaz

En el apartado 2.2 se estudiaron los tipos de aplicaciones disponibles (Android, iOS...). Sin embargo, uno de los requisitos era diseñar una aplicación compatible con cualquier dispositivo y adaptable a cualquier tipo de pantalla, y si bien era posible realizar varias aplicaciones para los distintos sistemas operativos, resultaba mucho más adecuado programar una aplicación en HTML5.

Además de realizar la programación en HTML5 también es de suma importancia utilizar estilos que sean correctamente visualizables en todos los dispositivos (Ordenador, tablet, smartphone...)

2.6.5. Elección del modo de fabricación de la carcasa

El diseño de la carcasa exterior es imprescindible para garantizar la durabilidad del producto y permite garantizar las condiciones de seguridad necesarias para el usuario.

Para la elaboración de la carcasa de los prototipos se estudiaron diversas opciones de fabricación: Moldeo en arena, moldeo a la cera perdida... pero finalmente se optó por utilizar una impresora 3D. Este dispositivo ofrece gran flexibilidad y permite realizar un prototipado rápido con bajos costes.

Para este proyecto se ha utilizado una Prusa Mendel con una densidad del 100 % y un ancho de capa de 0,4 cm. El material a utilizar es plástico ABS (Acrilonitrilo butadieno estireno) que es resistente y ligero.

La rapidez de obtención de piezas con la impresión 3D la hace ideal para la fase de diseño, donde ha sido necesario desarrollar distintas piezas hasta adaptar su funcionalidad. Sin embargo el acabado irregular, la baja exactitud y la necesidad de procesado final impiden que este proceso sea viable para la producción en serie, donde resultan mucho más adecuados métodos como extrusión o inyección de plástico.

Capítulo 3

Descripción del Hardware

En este capítulo se van a describir las especificaciones del hardware utilizado en el módulo estándar (Smartplug), los posibles sensores añadidos en los módulos especializados serán descritos en sus respectivos capítulos.

Figura 3.1: Conjunto de piezas del módulo estándar

3.1. Carcasa exterior

Como ya se mencionó en el capítulo anterior la carcasa se fabricará con una impresora en 3D de plástico, concretamente una Prusa Mendel con una densidad del 100 % y un ancho de capa de 0,4 cm. Así es necesario diseñar las piezas en un programa compatible con esta impresora.

3.1.1. Diseño: Catia

Actualmente existen una gran variedad de programas destinados al diseño de piezas 3D: Openscad, freeCad, SketchUp ... Habiendo estudiado las características individuales de cada uno se ha optado por utilizar Catia, un programa desarrollado por Dassault Systèmes destinado al modelado industrial y que permite una gran flexibilidad gracias a su entorno modular [18]. –

La gran cantidad de librerías y herramientas disponibles permiten satisfacer las necesidades del proyecto consiguiendo así un entorno de desarrollo óptimo.

Para el diseño individual de cada pieza se ha utilizado el módulo *part design*, que permite generar sólidos dando volumen a dibujos bidimensionales. De esta forma se simplifica el modo de trabajo. Una de las grandes ventajas de este modulo son las herramientas que contiene, cabe destacar *Hole* que permite crear talados de diámetro y profundidad deseada, o *Edge Fillet* que permite el redondeo de aristas con el radio deseado y que ha resultado imprescindible para lograr el acabado final.

3.1.2. Descripción de las piezas

Al tratarse de un primer prototipo se ha decidido utilizar para la pieza de los enchufes una regleta comercial, concretamente el enchufe modelo 9667776 de la marca Aki (figura 3.2), y diseñar el resto de piezas acorde a esa.

Figura 3.2: Enchufe modelo 9667776, desmontado y con separador adicional

A fin de encajar el resto de la electrónica en su interior ha sido necesario diseñar unas piezas auxiliares que permiten separar la parte superior de la inferior y crear un espacio intermedio perfecto para alojar tanto el Arduino, como los relés y los sensores. De esta forma se ofrece un producto final compacto y seguro, en el que el espacio se ha optimizado al máximo.

Además como en todo momento se busca conseguir un producto versátil capaz de adaptarse a las necesidades del usuario con el mínimo cambio, la carcasa exterior cuenta con una pieza intercambiable en función de los sensores que se deseen incluir.

La carcasa se halla dividida en 6 piezas independientes:

1. **Soporte de la clavija:** Es la parte inferior de la regleta comercial y su función es proporcionar alimentación a nuestro producto desde la red doméstica. Al tratarse de una pieza independiente resulta sencillo cambiarla para adaptar nuestro producto a otro tipo de conector u otras normativas eléctricas facilitando así la posibilidad de venta internacional.

2. **Soporte de los enchufes:** Es la parte superior de la regleta y permite conectar varios dispositivos eléctricos estándar de corriente alterna. En este caso se ha optado por permitir la conexión de 4 dispositivos, pero es posible ampliar o reducir las cantidad de enchufes presentes según las peticiones del usuario.

3. **Pared lateral:** Esta pared otorga mayor solidez al producto y sobre ella se situarán la fuente de alimentación y el reloj, además cuenta con unas aberturas cuadradas para permitir el paso de cables.

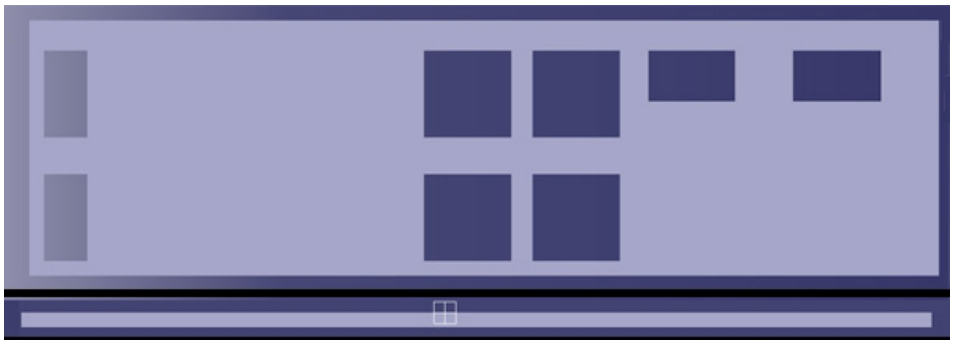


Figura 3.3: Pared exterior estándar

4. **Pared exterior estándar:** Tanto la parte exterior estándar como la variable se encargan de crear un espacio intermedio entre los dos soportes para alojar toda la electrónica en su interior, su diseño con raíles permite extraer todos los módulos en caso de avería.

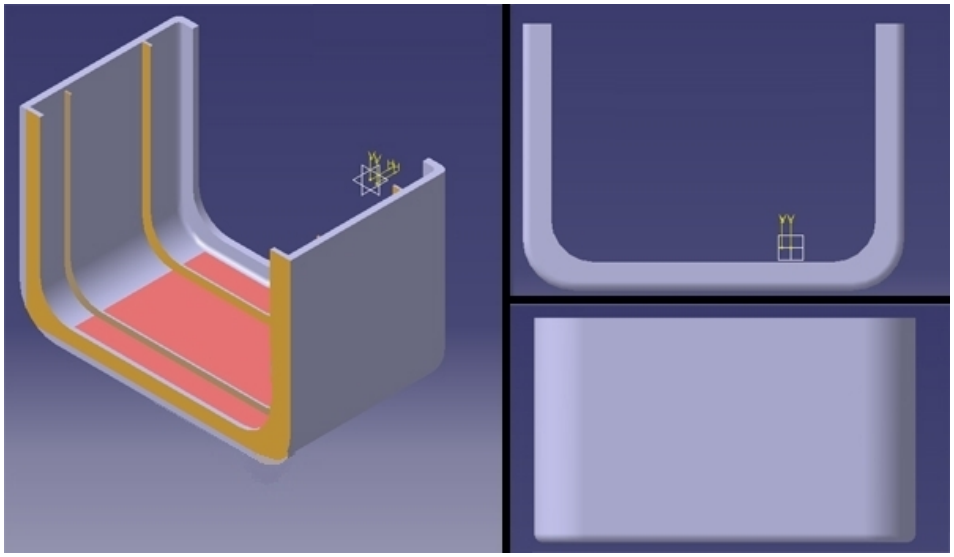


Figura 3.4: Pared lateral

5. **Pared exterior variable:** Esta pared contiene los sensores y dispositivos acústicos y/o luminosos de cada prototipo, por lo que existe un modelo diferente para cada combinación de sensores. Se han incluido dos pestañas exteriores a los lados sobre las que posteriormente se realizarán taladros, para poder unir de forma solidaria con un tornillo las dos paredes exteriores.

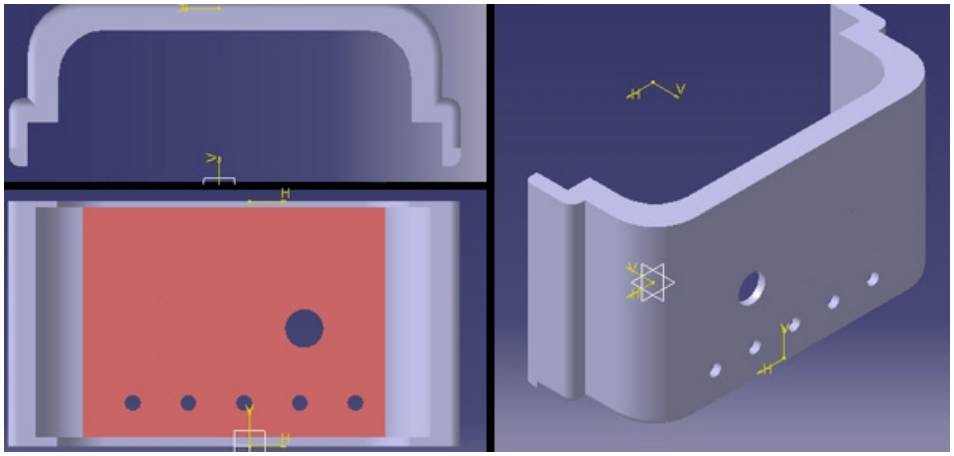


Figura 3.5: Pared exterior variable

6. **Soporte de electrónica:** Esta pieza es la encargada de fijar los relés y el Arduino ya que cuenta con cuatro soportes sobre los que atornillar estas placas. Además se han previsto unas ranuras de ventilación para evitar fallos por sobrecalentamiento al haberse observado cierta tendencia en el Arduino y en la fuente de alimentación. A fin de sujetar la pared lateral esta pieza cuenta con dos raíles sobre los que se situará, y en los extremos cuenta con dos hendiduras para permitir el paso de cables.

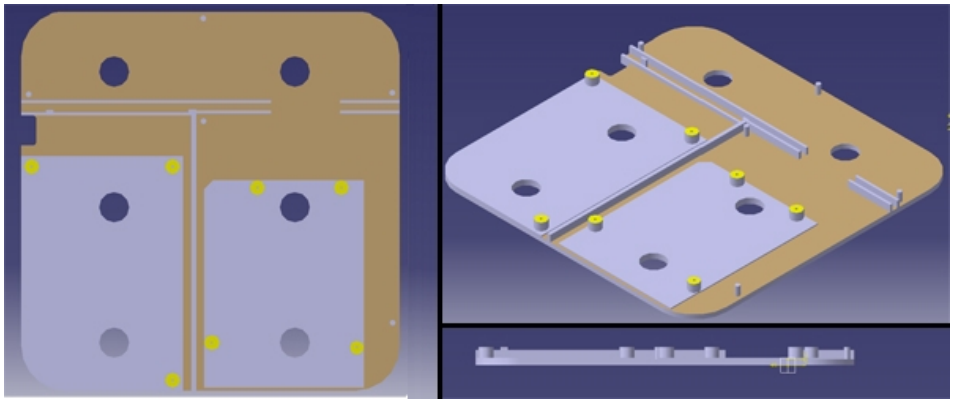


Figura 3.6: Soporte de electrónica

Cabe mencionar que el diseño de todas las piezas se ha visto afectado por el modo de fabricación que limita las formas de diseño.

Nota: Los planos de las piezas diseñadas en Catia con sus acotaciones se encuentran en el apéndice F.

3.1.3. Fallos de impresión y mejoras aplicadas

Tras una primera prueba de impresión se observaron ciertos factores que obligaron a realizar modificaciones en el diseño original y a ejecutar trabajos de acabado. Es importante tener en cuenta que se está utilizando una impresora doméstica y no una industrial por lo que la precisión y el acabado son peores.

Primero se observó que los taladros de menos de 2 mm de diámetro no eran realizables, por lo que se suprimieron y se optó por realizarlos manualmente.

También se comprobó que para imprimir las paredes exteriores era necesario girarlas 90º sobre el eje Z lo que causaba un peor acabado final y la necesidad de llevar acabo tratamientos de mejora. Finalmente se observo que todas las piezas impresas tenían tendencia a encoger, por lo que fue necesario revisar los diseños e incrementar algunas medidas.

Además al imprimir se observo que los diseños originales requerían algunos retoques, las columnas de atornillado eran demasiado finas y frágiles, la anchura del soporte de electrónica inferior era muy pequeña, y este necesitaba agujeros para la ventilación.

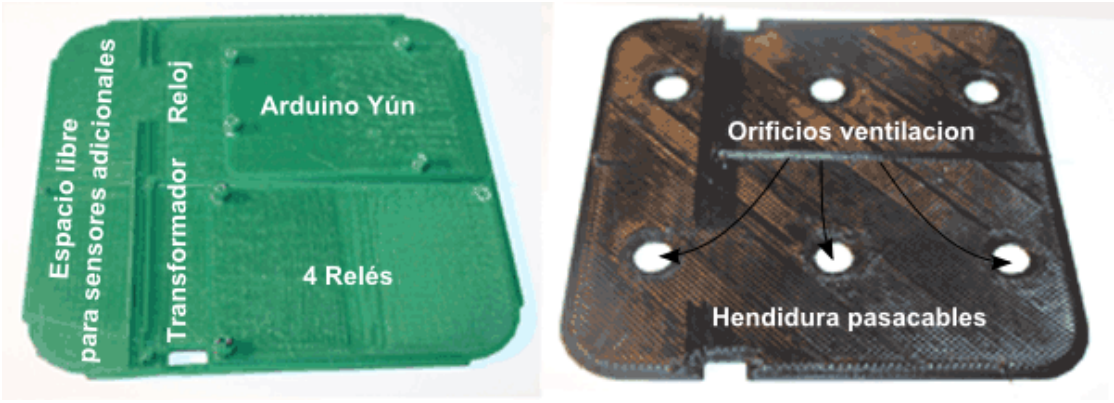


Figura 3.7: Modificaciones en la pieza del soporte

Tras corregir todos los factores que afectaron al diseño, sobre las piezas se llevaron a cabo los trabajos de acabado, se hicieron talados, se rellenaron huecos excesivos con masilla, se pintaron y lijaron las piezas.

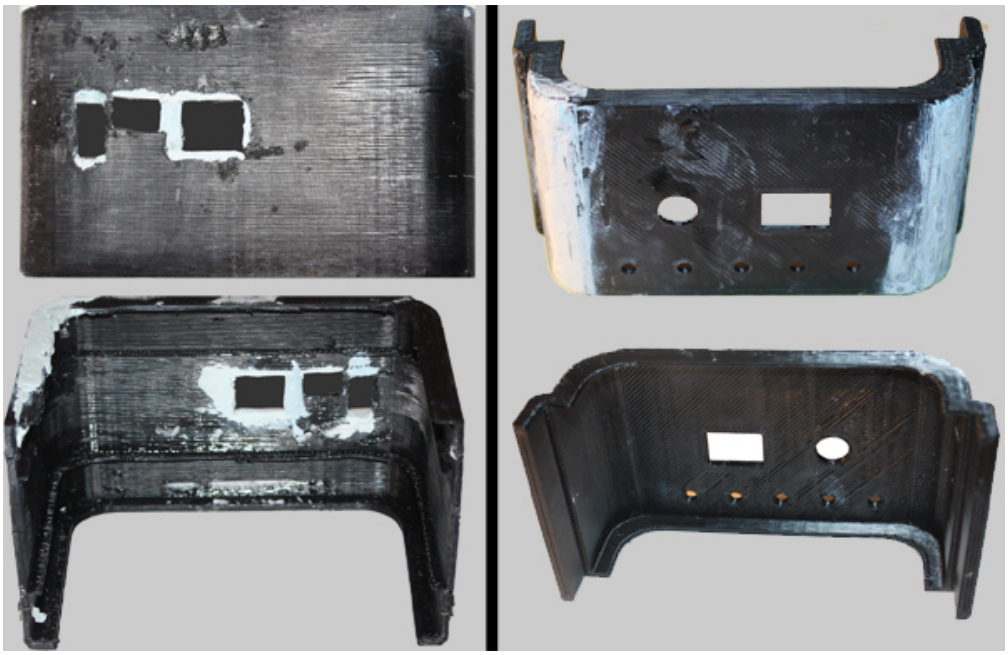


Figura 3.8: Carcasa exterior impresa y retocada

3.2. Fuente de alimentación

La función de la fuente de alimentación es convertir los 220V AC de la red eléctrica a la que se encuentra conectado el Smartplug a 5V DC para alimentar al Arduino. Se ha utilizado una fuente de alimentación comercial de la marca Hamma que cumple con la normativa de compatibilidad electromagnética EN 61000-3-2.

La alimentación puede realizarse a través del puerto microUSB o a través de los pines de alimentación Vin y GND. Como Arduino solo contiene un regulador interno en el puerto micro-USB si se desea utilizar los pines de alimentación es necesario proveer una fuente estable, para eso tras la fuente de alimentación se puede añadir un regulador externo como el LM7805. Sin embargo, aunque eso teóricamente funcionaría, cuando se llevó a cabo Arduino experimentó un comportamiento inestable, se reiniciaba continuamente y no era capaz de alimentar a los sensores que tenía conectados. Finalmente se decidió utilizar el microUSB y modificar el diseño original.

Figura 3.9: Fuente de alimentación 220-5V.

3.3. Relés

Un relé es un dispositivo electromecánico que funciona como un interruptor y permite controlar un circuito eléctrico de alta o media potencia [19].

Una de las principales ventajas que presenta, es la completa separación entre la linea eléctrica y el circuito de control, lo que resulta muy adecuado en nuestro caso para aislar y proteger el Arduino de las altas corrientes y voltajes.

Figura 3.10: Ywrrobot 4 relay board

En este caso se ha optado por utilizar una placa comercial, la *Ywrrobot 4 relay board* para controlar el estado de los cuatro enchufes. Esta placa puede ser controlada y alimentada directamente desde el Arduino, empleando los puertos GPIO y la salida de alimentación de 5V.

Los relés que utiliza la *Ywrrobot* son cuatro *songle SRD-05VDC-SL-C* que permiten operar en corriente alterna con intensidades de 10A y voltajes de entre 220V y 125V [20].

3.4. Reloj

Arduino cuenta con un reloj interno, sin embargo este carece de batería externa y solo funciona cuando el Arduino se encuentra conectado, por lo que solo puede medir incrementos de tiempo .

Para este proyecto es necesario un riguroso control de la fecha y hora, ya que se podrán programar los relés según intervalos horarios. Por ello se va a incorporar al circuito un reloj en tiempo real que permita contabilizar correctamente el tiempo. Concretamente se va a utilizar un circuito integrado con el reloj en tiempo real DS1387.

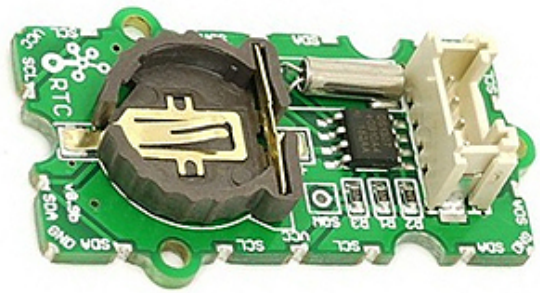


Figura 3.11: Reloj Tiempo Real DS1307

3.5. Unidad de Control: Arduino Yún

Arduino es un entorno de desarrollo abierto basado en distintas placas microcontroladoras (UNO, DUE, Leonardo, Yún, Nano, Mega, Freeduino, Teensy ...) que presentan una gran flexibilidad a bajo coste lo que las hace ideales para una gran variedad de proyectos.

Estas placas utilizan los microprocesadores: Atmel, AVR y principalmente los Atmega 328,1280, 2560. Tienen un entorno de desarrollo propio y se programan en Processing un C++ modificado.

Figura 3.12: Arduino Yún

Para este proyecto se ha optado por utilizar el Arduino Yún, una combinación del Arduino Leonardo (microcontrolador ATmega32U4) con el microprocesador Atheros linux AR9331. Así cuenta con la capacidad de desarrollo de un Arduino normal combinada con una máquina linux capaz de ejecutar scripts, de actuar como servidor, de acceder a una microSD y que le proporciona una conexión WIFI .

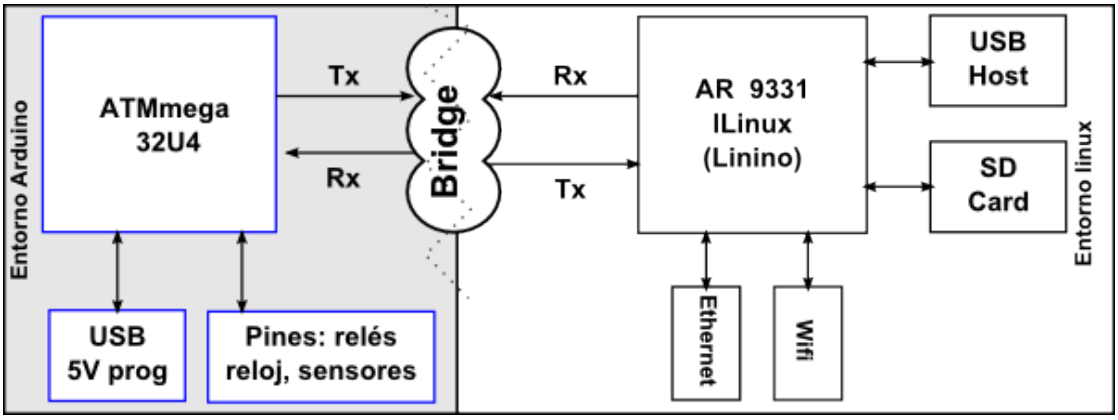


Figura 3.13: Arduino Yún esquema [15]

La comunicación entre ambos procesadores se realiza mediante la librería *Bridge*, pudiendo así enviar y recibir información entre ellos fácilmente. De esta forma el procesador linux recibe información de la tarjeta microSD, y del entorno web, mientras que el ATmega32U4 intercambia información con los GPIO.

Esta placa tiene disponibles 20 puertos de propósito general entre los cuales 7 pueden ser usados como salidas PWM y 12 como entradas analógicas. Cuenta con entradas USB, Ethernet y microUSB, y dispone de un lector de microSD [15].

3.5.1. Microcontrolador ATmega32u4

El microcontrolador ATmega32U4 opera a un voltaje de 5v y a una frecuencia de 16 MHZ, posee una memoria flash de 32KB, SRAM 2.5 kB, y EEPROM 1kB [15].

Solo el ATmega32U4 tiene acceso a los pines de entrada y salida. Estos pines operan a 5 voltios, pueden recibir un máximo de 40 mA y cuentan por defecto con una resistencia de pull-up de 20-50 kOhms.

Es necesario tener en cuenta que algunos pines tienen funciones especiales por lo que no es posible conectar algo a ellos si se desea utilizar estas funcionalidades.

Dado que existen multitud de funciones especiales solo se van a explicar aquellas que son relevantes para este proyecto. Hay que tener en cuenta que los pines 0 y 1 deben estar libres ya que se tratan de los pines de transmisión (0-RX) y recepción (1-TX) de la librería *Bridge*. Como ya se ha mencionado antes esta librería es la encargada de comunicar los dos microprocesadores por lo que resulta indispensable para nuestro proyecto.

Además los pines 2 y 3 GPIO son los encargados de transmitir la señales de reloj utilizando la librería *Wire*, de esta forma el 2 lleva la señal SDA, y el 3 la SCL.

Finalmente cabe mencionar que pines analógicos pueden ser usados como digitales, refiriéndose a ellos como pines desde el 18 al 24. Estos pines cuentan con un resolución de 10 bits, y toman las medidas con una referencia de 5V a tierra.

3.5.2. Microprocesador linux: Atheros AR9331

Este microprocesador tiene una frecuencia de reloj de 16 MHz, su arquitectura es MIPS 400 MHz y opera a un voltaje de 3.3V. La RAM es de 64 MB, y la memoria flash tiene un espacio de 16 MB [15].

De la memoria flash 9 MB se encuentran ocupados con una distribución de linux llamada OpenWrt-Yún (Linino) basada en OpenWrt. Esta distribución puede ser modificada de varias maneras: instalando y descargando librerías, accediendo a los distintos ficheros de configuración... aunque siempre es posible restaurar la configuración inicial utilizando los botones de reset.

Finalmente el microprocesador de Linux tiene acceso a un USB tipo A, al lector de tarjetas microSD, la entradas Ethernet y puede actuar como servidor.

3.5.3. Comunicación: Bridge

La librería *Bridge* permite comunicar de forma sencilla los dos procesadores del Arduino Yún. Las sentencias enviadas por el ATmega32U4 gracias al *Bridge* son interpretadas por Python en el AR9331. Así es posible ejecutar programas en el Linux desde el ATmega32U4 y almacenar datos obtenidos por los sensores en la microSD.

De la misma forma esta librería permite enviar datos del AR9331 al ATmega32U4, así puede recibir comandos por el entorno web (AR9331), procesarlos y enviarlos al ATmega32U4 para que este los ejecute, o programar el microcontrolador utilizado la WIFI.

Además la librería *Bridge* es la que permite utilizar la consola a través de la WIFI, lo que resulta muy útil en la etapa de desarrollo.

Capítulo 4

Módulo estándar

El módulo estándar es una pieza compuesta por toda la electrónica, programación y partes de carcasa comunes a todos los módulos, es totalmente independiente y está encargada de las funcionalidades generales. Esto permitirá una fabricación en serie, reduciendo así los costes, y los tiempos de desarrollo. Además de la posibilidad ofrecerla como kit de desarrollo de nuevos productos.

En este proyecto el módulo estándar esta compuesto por distintas placas comerciales. Lo ideal sería desarrollar una placa que contuviera toda la electrónica estándar y que estuviera totalmente adaptada a este producto. Esto permitiría reducir el tamaño y los costes al eliminar todas las funciones que ofrecen estas placas que para este proyecto resultan innecesarias, y lograr un mayor aprovechamiento del espacio, aunque esta integración es parte de las mejoras propuestas y se realizará en el futuro.

En este módulo se encuentran:

1. Clavija
2. Regleta de cuatro enchufes
3. Placa de Relés
4. Fuente de alimentación
5. Reloj en tiempo real
6. Zumbador
7. Leds de señalización
8. Arduino Yún

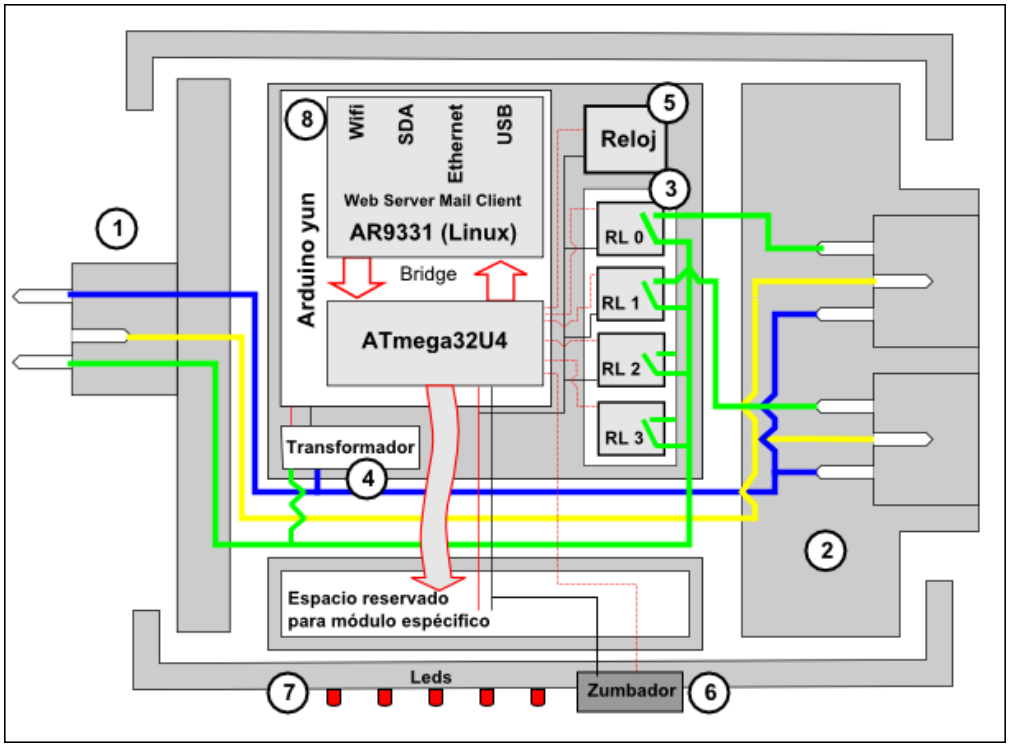


Figura 4.1: Esquema del módulo general

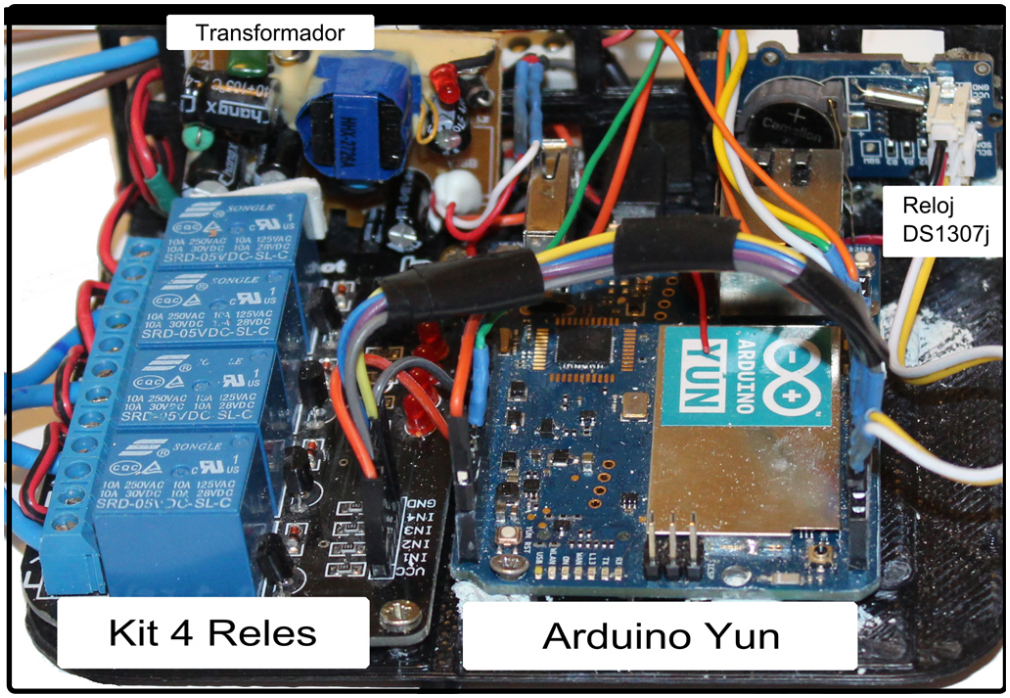


Figura 4.2: Módulo general

Posteriormente se procederá a describir las conexiones de forma más detallada.

4.1. Clavija

Su función es alimentar a todos los elementos del Smartplug de esta forma se evita el uso de baterías que requieren más mantenimiento. Además cabe mencionar que la clavija es una pieza totalmente independiente, lo que permite adaptar el kit a las distintas normas internacionales de electricidad.

Conexiones

La clavija debe estar conectada a la fuente de alimentación, a la placa de relés y a los enchufes superiores.

El borne de la clavija 1 esta conectado a la fuente de alimentación y a los cuatro relés en su entrada NA. Las tomas de tierra de la clavija y la regleta de enchufes están unidas entre sí (2). Finalmente el otro borne de la clavija (3) esta conectado a la fuente de alimentación y a cada uno de los enchufes de la regleta.

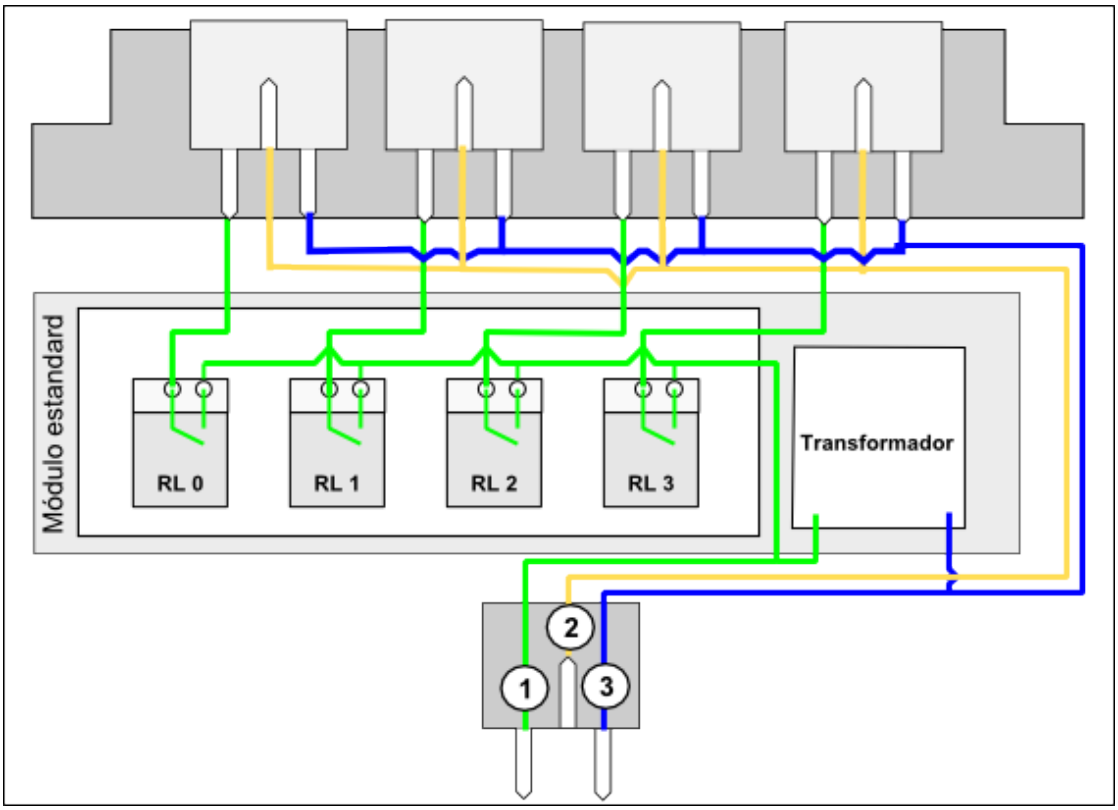


Figura 4.3: Esquema de conexiones relacionadas con la clavija

4.2. Regleta de cuatro enchufes

Su función es alimentar a cuatro dispositivos eléctricos como pueden ser: un ventilador, una lampara, un calefactor, un motor...

Conexiones

Cada uno de los cuatro enchufes tiene conectado directamente desde la clavija una de las fases y la tierra. La otra fase se encuentra conectada al relé que lo gobierna. Para más detalle consultar apartado. 4.3

4.3. Placa de relés

Su función es controlar el estado de los enchufes según las señales enviadas desde el Arduino.

Conexiones

La placa de relés debe ser alimentada a través de la salidas de 5V y GND del Arduino. Además utilizará los pines 4-7 para controlar el estado de los relés, de esta forma los conectores marcados como IN1, IN2, IN3 y IN4 deben ser conectados a los pines 4-7 del Arduino.

Cada uno de los relés de la placa cuenta con una clema con tres posibles conexiones:

1. **Central o COM:** Es común para los dos modos de funcionamiento del relé. Los relés tienen esta salida unida entre si, y conectada a un borne de la clavija.
2. **Normalmente Abierto (NA):** Si se encuentra en esta posición los relés por defecto están desconectados. En este proyecto se va utilizar esta conexión, así cada uno de los relés tendrá conectada esta salida a su respectivo enchufe superior.
3. **Normalemente Cerrado (NC):** En esta posición los relés se encuentran por defecto conectados. En este proyecto no se utiliza.

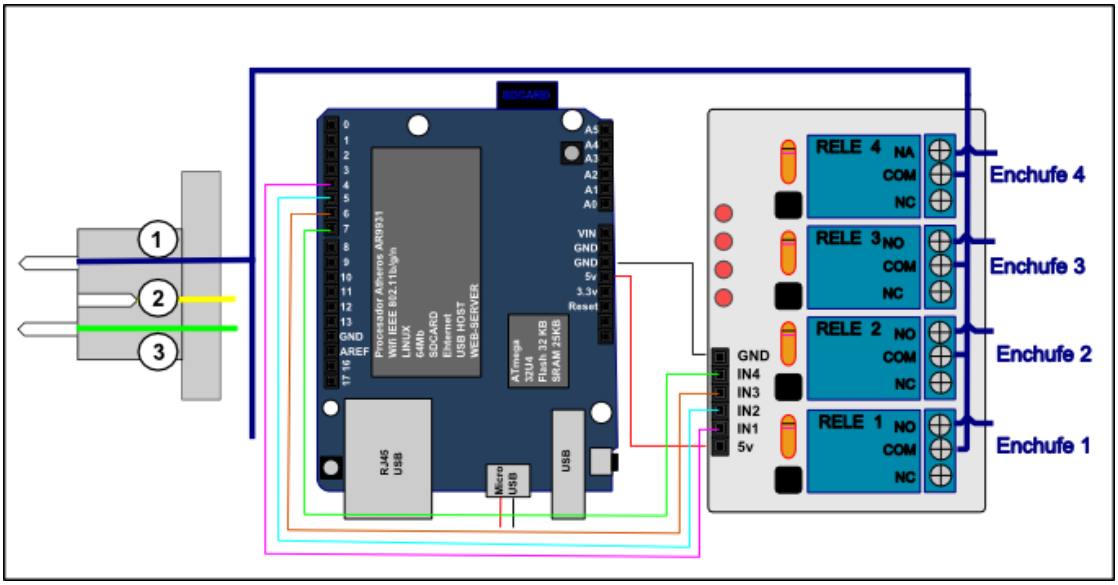


Figura 4.4: Esquema de conexiones relacionadas con la placa de relés

4.4. Fuente de alimentación

Su función es tomar la alimentación desde la clavija a 220V en alterna y transformarla a 5 V en continua.

Conexiones

Recibe la alimentación desde la clavija, y la transforma a 5 V para alimentar al Arduino por el puerto microUSB.

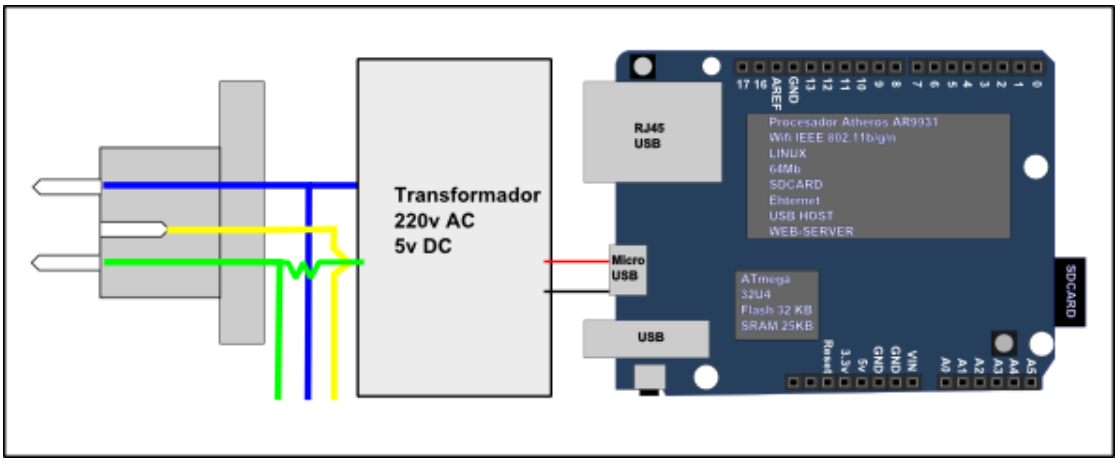


Figura 4.5: Esquema de conexiones relacionadas con la fuente de alimentación

Nota: En un primer momento se optó por alimentar el Arduino por los pines de alimentación Vin y GND. Sin embargo, este es un pin no regulado que necesita una alimentación a 5V exactos fallando ante mínimas oscilaciones. Se intentó incluso utilizar una fuente de alimentación y un regulador LM7805, pero Arduino es muy sensible en estos pines y fallaba con gran frecuencia. Por esta razón finalmente se optó por alimentar el Arduino a través del puerto microUSB donde se obtenían buenos resultados.

4.5. Reloj en tiempo real

Su función es proporcionar la hora y fecha actual independientemente de que el dispositivo este conectado a Internet. El reloj se utiliza para el modo de programación de los relés en intervalos horarios.

Conexiones

La alimentación se realiza conectando las salidas de 5V y GND procedentes de la placa Arduino Yún a los respectivos pines módulo del reloj.

Los pines de comunicación son SDA y SLC que se conectan respectivamente a los pines digitales 2 y 3, estos pines son los reservados por la placa Arduino Yún para las comunicaciones I^2C y no pueden ser cambiados.

Nota: Aunque el Linino está configurado para obtener la hora por medio del protocolo de Internet NTP (Network Time Protocol), no hemos elegido esta opción para permitir usar el kit sin tener la necesidad de estar conectado a Internet.

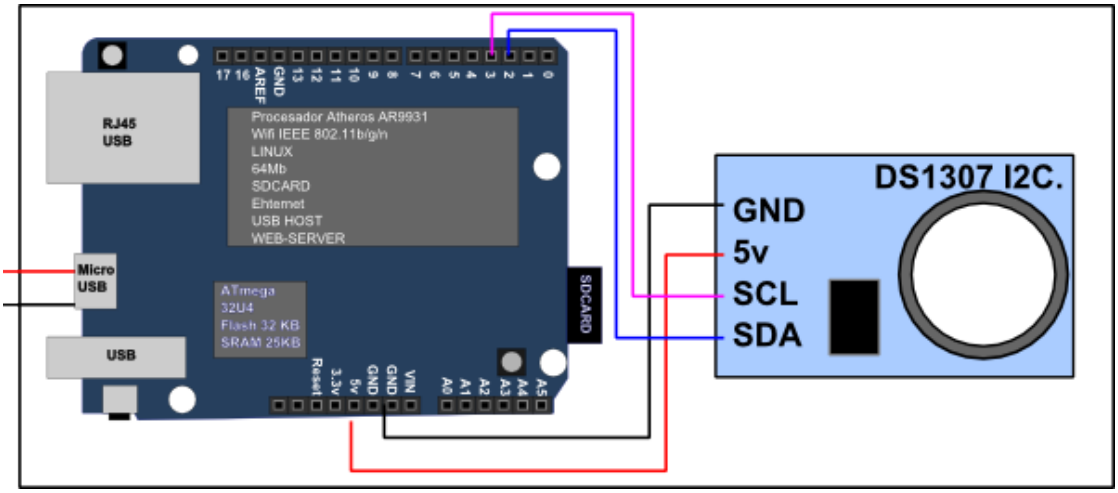


Figura 4.6: Esquema de conexiones relacionadas con el reloj

4.6. Zumbador

Su función en el módulo estándar es indicar al usuario cuando el Smartplug se encuentra listo tras enchufarlo. El zumbador suena brevemente al enchufar el Smartplug para indicar que está correctamente conectado, después suena otra vez cuando el dispositivo se encuentra preparado para operar.

En los módulos especializados el zumbador adquiere otras funciones, como ante la detección de gas ejecutar señales de alarma .

Conexión

El terminal positivo del zumbador se encuentra conectado al pin digital de Arduino 8, y el terminal negativo a la salida de Arduino GND.

4.7. Leds de señalización

Su función es indicar visualmente al usuario el estado de funcionamiento del Smartplug sin necesidad de consultar la Web.

El led de estado del Smartplug, es decir el primero, al conectar el dispositivo a la corriente se enciende brevemente permaneciendo apagado hasta que la inicialización de todo el software está completa, momento a partir del cual quedará permanentemente encendido.

Los leds de señalización de los relés son los cuatro siguientes, iluminados indican que el relé está conectado y viceversa.

Conexión

Estos leds utilizan los pines de Arduino A0-A4, aunque estos pines son analógicos pueden ser utilizados como pines digitales refiriéndose a ellos en el código como pines del 18-22.

De esta forma el led de estado del Smartplug utiliza el pin A0 (18), y los relés los pines A1-A4 (19-22).

4.8. Arduino Yún

El Arduino es la unidad de control. Su función es procesar la información recibida desde el entorno web y desde los dispositivos conectados a él, y ejecutar los procesos pertinentes.

Como ya se ha mencionado está constituido por el microcontrolador ATmega32U4 y el microprocesador AR9331 con una imagen de Linino (una distribución de linux para microcontroladores).

Las comunicaciones web dependen del microprocesador AR9331, el microprocesador de linux que utilizando la red WIFI recibe información de las páginas web y la procesa para enviarla utilizando el Bridge al microcontrolador ATmega32U4. Este microcontrolador se encarga de ejecutar en los relés y en el reloj las ordenes recibidas de acuerdo con la programación.

Además el Arduino Yún tiene conectada una tarjeta microSD en la que se almacenan las páginas web y a la que solo tiene acceso el microprocesador linux.

4.8.1. Resumen de conexiones

Es importante tener en cuenta que los pines digitales del 0-8 y los analógicos del A0-A4 quedan reservados para uso exclusivo de este módulo, dejando el resto libres para los módulos especializados.

Nº de Pin	Función
0	Debe ser dejado libre para utilizar las funciones del Bridge.
1	Debe ser dejado libre para utilizar las funciones del Bridge
2	Entrada/Salida SDA del reloj
3	Entrada/Salida SCL del reloj
4	Salida placa de relés IN1.
5	Salida placa de relés IN2
6	Salida placa de relés IN3
7	Salida placa de relés IN4.
8	Zumbador.

Tabla 4.1: Resumen de los pines digitales en el módulo estándar

Nº de Pin en la placa	Nº Pin funcionando como digital	Función
A0	18	Led de estado del Smartplug
A1	19	Led ligado al estado del relé 1
A2	20	Led ligado al estado del relé 2
A3	21	Led ligado al estado del relé 3
A4	22	Led ligado al estado del relé 4

Tabla 4.2: Resumen de los pines analógicos utilizados como digitales

4.8.2. Configuración de red inicial del Arduino

Por defecto Arduino viene de fábrica configurado como un punto de acceso WIFI. Esto no siempre resulta adecuado ya que no permite acceso a Internet, y por lo tanto algunas funcionalidades no se encuentran operativas (Ej: mandar emails). En caso de necesitar estar conectado a Internet se debe configurar el Arduino como un cliente dentro de una red.

Pasos para configurar la conexión Arduino como cliente DHCP

Paso 1 Acceso al menú de configuración: Utilizando el panel de configuración web de Arduino (consultar apéndice C) acceder al menú de configuración de redes, concretamente al apartado *WIRELESS PARAMETERS* (figura 4.7)

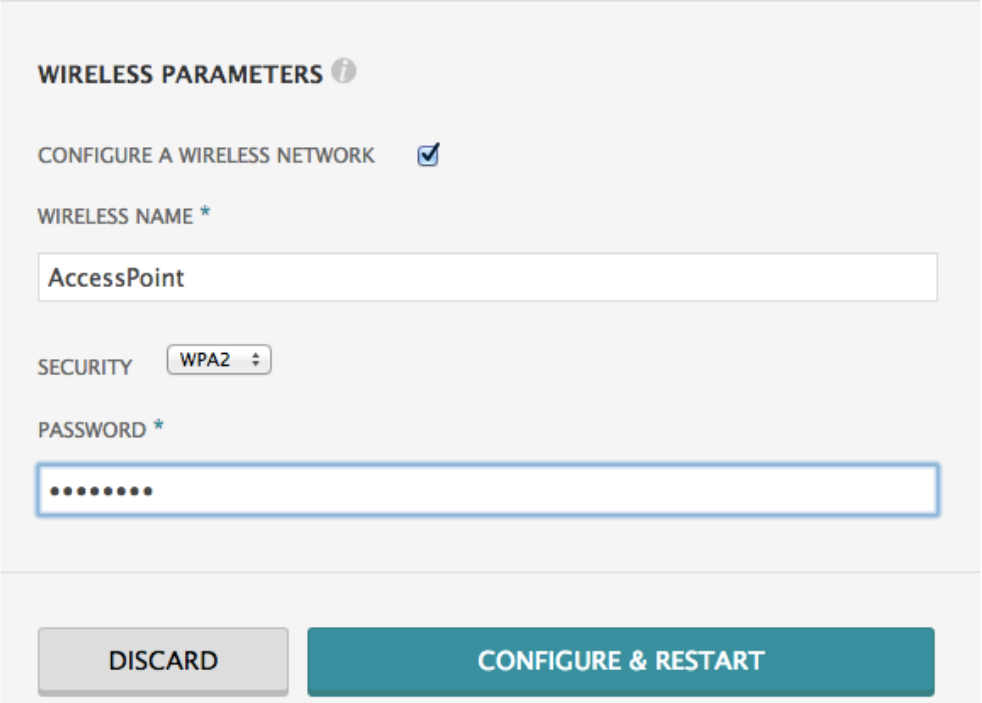


Figura 4.7: Menú de configuración de redes (Paso 1)

Paso 2 Introducir datos de red: En el apartado *WIRELESS PARAMETERS* hacer click en recuadro *CONFIGURE A WIRELESS NETWORK*, después introducir el nombre de la red y la contraseña en sus respectivas casillas, y finalmente en la opción *REST API ACCESS* seleccionar open.

Paso 3 Ejecutar los cambios: Una vez introducidos los datos solo es necesario hacer click en *CONFIGURE Y RESTART*. Arduino tarda unos minutos en establecer estos cambios, por lo que aparece una barra de tiempo como la de la figura 4.8 para controlar el avance. Una vez trascurrido este tiempo debemos reiniciarlo para que el Arduino pueda ejecutar los cambios.

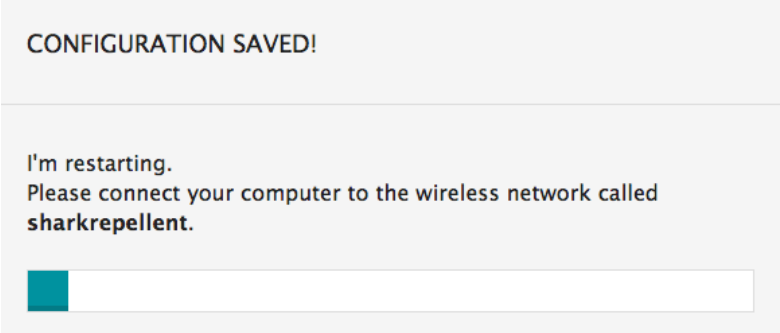


Figura 4.8: Establecer cambios en las conexiones (Paso 3)

Paso 4 Encontrar la nueva dirección del Arduino: Si todo ha ido correctamente, al conectar Arduino y consultar una herramienta exploradora de redes WIFI ya no aparecerá la red : *Arduino Yún-XXXXXXXXXXXX*.

Para encontrar la nueva dirección del Arduino se debe utilizar un software escaner de redes TCPIP como el IpScan23 (consultar apéndice E).

Nota: En caso de error es necesario resetear la red WIFI del Arduino a su valor por defecto, es decir a punto de acceso. Para ello hay que pulsar el botón *WIFI Reset* durante 15 segundos. Después se puede repetir de nuevo el proceso.

Nota: Si la configuración es errónea o no tiene acceso a la red que se ha configurado, después de varios intentos fallidos Arduino resetea automáticamente su configuración de red, y vuelve a su estado original como punto de acceso.

4.8.3. Corrección manual de bugs de Arduino

Como ya se ha mencionado antes, se está utilizando la versión Beta del IDE de desarrollo del Arduino Yún (ya que es la única disponible que soporta esta placa), por lo que aún existen ciertos fallos que afectan a su correcto funcionamiento. Estos problemas están reportados en la página oficial de Arduino, y los fabricantes ofrecen soluciones provisionales para corregirlos en esta versión.

Concretamente los fallos que afectan al funcionamiento de este proyecto están relacionados con el *Bridge*. El primero que se ha detectado es el fallo al enviar cadenas de más de 100 caracteres, y el segundo es un fallo en el arranque del *Bridge* al conectarse el Arduino.

Cabe mencionar que las soluciones ofrecidas son simplemente "parches"para conseguir un buen funcionamiento en esta versión, y que en versiones posteriores de esta placa no serán necesarias.

Advertencia: Las soluciones descritas a continuación consisten en modificar ficheros de configuración internos, por lo que es necesario seguir los pasos atentamente para evitar dañar o perjudicar el funcionamiento del Arduino.

Guía para corregir el Fallo al enviar cadenas de más de 100 caracteres

Al enviar cadenas de más de 100 caracteres por el *Bridge* a menudo se producen fallos en las comunicaciones, produciéndose resultados inesperados como: cadenas entregadas incompletas, o que el Arduino deje de responder.

Sin embargo esta limitación no es física, sino que viene impuesta por dos ficheros de configuración de Arduino por lo que puede ser corregida simplemente modificándolos.

Paso 1: Utilizar un cliente SFTP como el programa WinSCP para conectarse al Arduino, este programa permite la comunicación SFTP con un entorno gráfico fácil de utilizar. Consultar apéndice B para más información.

Paso 2: Es necesario modificar el fichero *Arduino /usr/bin/run-Bridge.py*

Para realizar esta modificación debe descargar el archivo a su Pc y hacer una copia de seguridad en él. Después con un editor de texto plano buscar la linea que pone:

```
PID=‘/usr/bin/pgrep -f "python Bridge.py"
```

y sustituirla por la siguiente :

```
PID=‘/usr/bin/pgrep -f "python -u Bridge.py"
```

Finalmente utilizando de nuevo el WinSCP reemplazar el archivo original en el Arduino.

Paso 3: Editar el fichero */usr/bin/kill-Bridge*, buscando la linea:

```
exec python Bridge.py 2$>$ /tmp/Bridge.py-stderr.log
```

y reemplazándola por:

```
exec python -u Bridge.py 2$>$ /tmp/Bridge.py-stderr.log
```

Para realizar esta modificación repetir el mismo procedimiento que se ha seguido en el paso anterior.

Guía para corregir el fallo en el arranque del Bridge al resetearse el Arduino.

Este fallo está provocado por la falta de sincronía en el arranque de los dos microprocesadores. Al conectarse el Arduino, el microcontrolador ATmega32U4 arranca mucho más rápido que el procesador de linux, esto produce que el ATmega32U4 trate de acceder al *Bridge* antes de que el AR9331 esté listo , por lo que el *Bridge* es incapaz de iniciarse correctamente y en muchos casos se bloquea.

Para solucionar este problema no es necesario modificar ficheros de configuración, sino que basta con añadir una espera de 30 segundos delante de la sentencia *Bridge.begin ()* en el código del ATmega32U4. Esto garantiza que ambos procesadores estén listos al iniciarse el *Bridge*.

4.8.4. Entorno de desarrollo

Este proyecto se ha realizado en un entorno Microsoft Windows 7 en el que se ha instalado:

1. **IDE 1.5.7 Beta de Arduino:** Para la programación relativa al ATmega32U4, utilizando C++.
2. **Notepad ++:** Un editor de texto plano para las páginas web y los scripts de Python 2.7.
3. **Putty:** Para la instalación de librerías adicionales en Linino.
4. **WinSCP:** Para la transmisión de archivos por SFTP (Secure File Transfer Protocol).
5. **Ipscan23:** Para localizar la dirección IP del Arduino.

4.8.5. Software desarrollado en relación al ATmega32U4

El ATmega32U4 contiene toda la programación relacionada con el control de los dispositivos conectados a los pines del Arduino. Es decir, se encarga de procesar las ordenes recibidas a través del *Bridge*, y ejecutarlas en los distintos dispositivos (reloj y relés en el módulo general). Además se encarga de leer la información de los dispositivos y transmitirla utilizando el *Bridge* al entorno Web que corre en el procesador AR9331.

Es necesario tener en cuenta que la memoria de este procesador es realmente pequeña, por lo que se debe optimizar su uso, y transferir las funciones que sean posibles al otro procesador que es de mayor tamaño.

Además se ha observado que este compilador no realiza bien las operaciones de liberación de memoria, por lo que hay que reducir al máximo la creación de variables y objetos temporales. Así, aunque lo ideal sería que cada dispositivo contara con una clase separada para su programación, esto solo se ha realizado en los relés que tienen la mayor cantidad de funciones propias.

4.8.5.1. Estructura de los ficheros y carpetas

Los ficheros de desarrollo de software para el ATmega32U4 siguen siempre el mismo esquema:

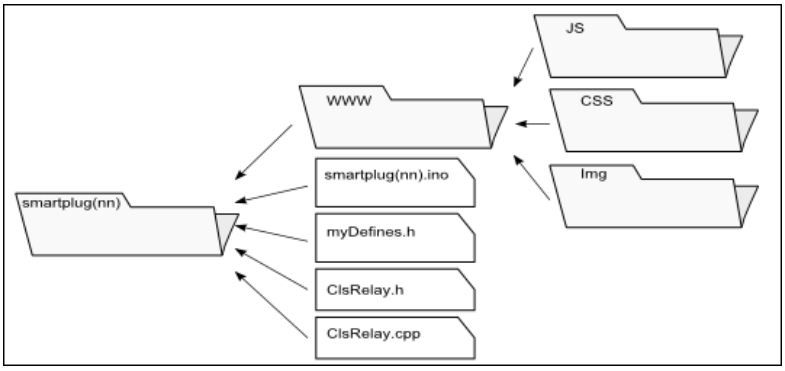


Figura 4.9: Esquema general de organización de ficheros y carpetas en el ATmega32U4

Todos los archivos se encuentran en una carpeta con el nombre del programa (Smartplug (nn)), donde nn indica la versión. Esta carpeta contiene los ficheros que componen el software del ATmega32U4 (mydefine.h, smartplug(nn).ino, clsRelay.h y clsRelay.cpp) y la carpeta *www* .

La carpeta *www* contiene los ficheros relacionados con la página web divididos en tres capetas (*css*, *js* y *img*) según su extensión de archivo. Aunque la carpeta *www* contiene la programación relacionada con el procesador linux es necesario situarla dentro de la carpeta Smartplug para que al subir el programa .ino suban automáticamente los archivos de las páginas web al Arduino.

4.8.5.2. Estructura del programa .ino y librerías

Los programas de Arduino se estructuran en cinco secciones: la primera encargada de declarar las librerías, la segunda las variables globales, la tercera el bloque *setup()*, la cuarta el bucle *loop()* y la última las funciones definidas por el usuario.

A continuación se encuentra el diagrama de flujo del programa .ino con sus bloques de ejecución principales y su relación con las funciones más relevantes. Cabe mencionar que algunas de las funciones definidas por el usuario no se encuentran en el diagrama ya que son llamadas desde el interior de otras.

Este diagrama es común a todos los módulos ya que el funcionamiento general del programa siempre es igual y las modificaciones de software que se llevan a cabo para desarrollar módulos especializados se realizan en el interior de las funciones.

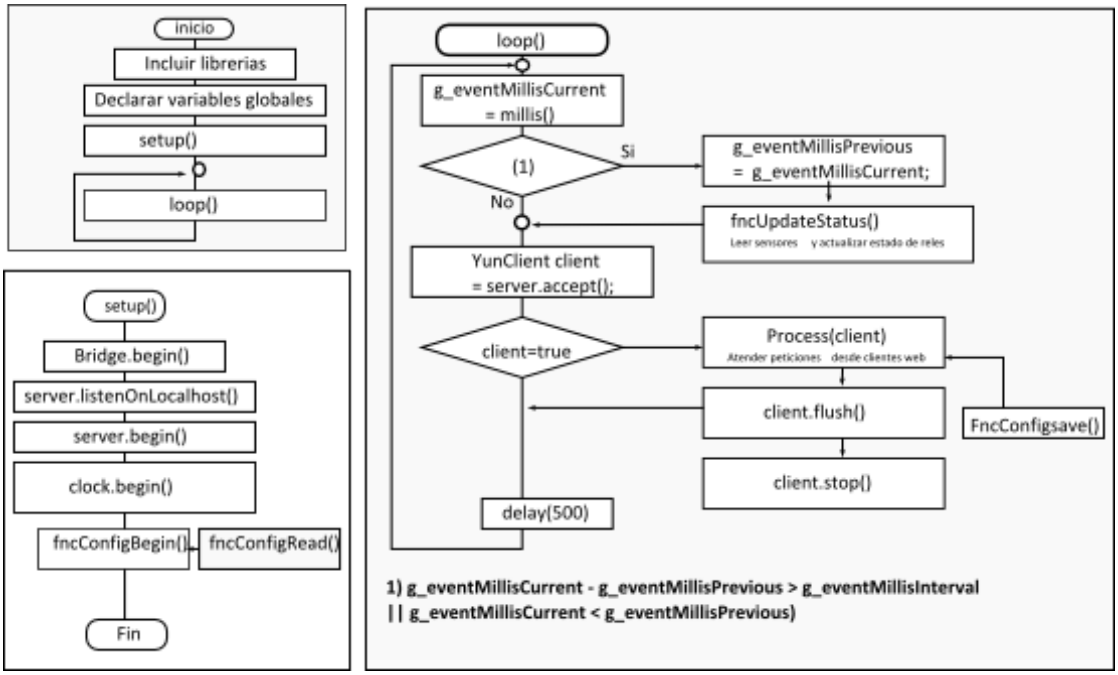


Figura 4.10: Estructura general del programa

Descripción de las secciones del programa .ino de Arduino

1. **Librerías utilizadas:**

   a) **Bridge.h:** Permite la comunicación entre los procesos que se llevan a cabo en el microcontrolador ATmega32U4 y en el microprocesador AR9331.

   b) **YúnServer.h:** Permite recibir en el microcontrolador ATmega32U4 las peticiones enviadas desde el entorno web, que llegan utilizando el *Bridge*.

   c) **YúnClient.h:** Permite enviar desde el ATmega32U4 datos al entorno web utilizando el *Bridge*.

   d) **FileIO.h:** Permite escribir y leer ficheros en la microSD. Realmente es el microprocesador linux el que realiza la lectura y escritura y los datos se envían utilizando el *Bridge*.

   e) **Wire.h:** Permite la comunicación con dispositivos que utilizan el protocolo de comunicación I^2C.

   f) **DHS1307.h:** Facilita el uso del reloj en tiempo real.

   g) **ClsRelay.h:** La clase relé es de desarrollo propio, y se encarga de todas las funciones relacionadas con el uso y programación de los relés.

2. **Variables globales** Se han utilizado un exceso de variables globales en este programa. Esto es debido a que la inicialización de variables dentro

de funciones cuando se está al límite de ocupación de la memoria, causa problemas e inestabilidad.

Todas las variables se encuentran explicadas dentro del código fuente, no obstante, cabe resaltar las siguientes:

a) **g_BufferSTD:** Es una matriz que contiene toda la información del módulo estándar, su función es guardar la configuración y estado en la microSD.

b) **g_BufferMOD:** Espacio reservado para guardar la configuración y estado de los módulos específicos.

c) **g_BufferWEB:** Utilizado para enviar toda la información del estado y configuración del Smartplug mediante ajax a los clientes web.

d) **g_oRL:** Us una matriz compuesta por cuatro objetos de la clase Relé.

e) **g_Clock:** Es un objeto derivado de la clase DS1307.

3. **Setup():** Esta función se ejecuta solo una vez al iniciarse el Arduino. En ella se incluye: la inicialización de procesos y objetos, la lectura de la última configuración guardada en la microSD, y la asignación de pines.

   Los procesos que deben iniciarse son: los relacionados con el *Bridge*, los del servidor y los del reloj. Para ello al principio del programa se deben añadir las sentencias:

```
delay(500);
bridge.begin();
server.listenOnLocalhost();
server.begin();
clock.begin();
```

   Nota: El *delay(n)* que se ha añadido previo al *Bridge.begin()* es imprescindible para evitar que se den problemas con el *Bridge*. Además en caso de añadirse servicios adicionales como enviar emails o programas python, es necesario aumentarlo hasta valores entorno a 2000.

   Tras inicializar los procesos del Arduino debe cargarse la última configuración guardada en la microSD en las variables globales. Para ello se debe llamar a la función:

```
fncConfigBegin();
```

   En cuanto a la asignación de pines deben declarase los puertos del Arduino que van a utilizarse y definir si son puertos en entrada o salida. Esto se realiza con estas dos sentencias:

```
pinMode(Numero de Pin, INPUT)  // Entrada
pinMode(Numero de Pin, OUTPUT) // Salida
```

4. **Loop():** Esta función se encarga de los procesos que deben ejecutarse repetidamente como: atender a las peticiones enviadas por los clientes web, leer el estado de los dispositivos y actualizar el estado de los actuadores en función de las lecturas y modo programación escogido.

   El bucle *loop()* consume mucha energía ya que mantiene al microcontrolador continuamente en funcionamiento. A fin de ahorrar energía y evitar sobrecalentamientos se ha optado por repetir las sentencias de actualización cada cierto intervalo de tiempo, concretamente un segundo.

   Esto se ha logrado introduciendo una sentencia al comienzo del bucle *loop()* y utilizando el reloj interno del Arduino como contador. De esta forma al comienzo del bucle se compara el valor contador con el periodo estipulado y solo cuando es mayor se reinicia el contador y se realizan los procesos, en este caso se ejecuta la función *fncUpdateSts()* encargada de las actualizaciones.

   Finalmente en todos y cada uno de los bucles se comprueba si existe un nuevo cliente web y en caso de existir, se ejecuta la función *Process()* que procesa y ejecuta las ordenes. Más tarde se procederá a explicar con detalle esta función.

   Así el bucle *loop()* queda de la siguiente forma:

```
void loop(){
  //Lectura del reloj interno Arduino
  g_eventMillisCurrent = millis();
  if (g_eventMillisCurrent - g_eventMillisPrevious >
   g_eventMillisInterval || g_eventMillisCurrent <
   g_eventMillisPrevious)
  {
    g_eventMillisPrevious =  g_eventMillisCurrent;
    fncUpdateStatus();
    // Escuchar peticiones web
    YunClient client = server.accept();
    if (client) {
      Process(client);
      // Envía todo el buffer antes de cerrar
      client.flush();
      // Cerrar conexiones web (máximo 5 clientes)
      client.stop();
    delay(500); // Un pausa para estabilizar.
  }
}
```

5. **Funciones definidas por el usuario:** Se han definido las funciones: *Process()*, *fncConfigSave()*, *fncConfigBegin()*, *fncRead*, *fncUpdateStatus()*, *fncFillStatus()*. Estas funciones se han creado para realizar distintas operaciones del Smartplug, más tarde se procederá a explicarlas en detalle.

4.8.5.3. Clase Relé

Como ya se ha dicho previamente los relés componen una clase independiente. Se utilizan en el código principal dentro del las funciones *fncUpdateStatus()*, y *Process()* y se encargan de todo lo relacionado con la programación de los relés.

Modos de programación

El módulo estándar dispone de tres modos de programación de los relés:

1. **PRGMANUALON:** La programación manual on conecta el relé independientemente de cualquier otro tiempo de información recibida. En el código para ahorrar memoria este modo ha sido denominado modo 1.
2. **PRGMANUALOFF:** La programación manual off desconecta el relé. En el código denominado modo 0.
3. **PRGAUTOCLOCK :** Permite programar los relés para que se conecten fuera o dentro de un intervalo horario. En el código denominado modo 2.

Nota: En los módulos especializados existen modos de programación adicionales en relación a la información recibida por sus sensores.

Atributos propios

Cada relé cuenta con unos atributos propios que determinan su modo de funcionamiento:

1. **byte g_pin** : Pin de Arduino al que se encuentra conectado.
2. **byte g_pinLed**: Pin del led ligado al estado (conectado, desconectado) en que se encuentra el relé.
3. **byte g_stdDef**: Variable que indica modo de funcionamiento por defecto, en este proyecto por defecto se encuentran desconectados.
4. **g_bPrgMode** : Modo de programación escogido (consultar 4.8.5.3)
5. **g_iprgSlotStart**: Minuto del día de comienzo del intervalo para la programación en modo 2.
6. **g_iPrgSlotEnd**: Minuto de día de fin del intervalo para la programación en modo 2.
7. **g_bSlotModeIn**: Esta variable determina si se debe conectar el relé dentro del intervalo señalado o fuera de él. Un cero indica que el relé permanecerá conectado fuera del intervalo marcado para este relé y un uno indica que permanecerá conectado dentro.

Funciones

La funciones ligadas a la clase relé son las siguientes :

1. **begin(unsigned short bPinRelé, unsigned bPinLed)** encargada de asignar el pin del relé y el led asociado a este. Además asigna los valores por defecto a las variables correspondientes a : modo de programación, minuto de inicio del intervalo, minuto de fin del intervalo y al estado de funcionamiento por defecto (dentro o fuera del intervalo), estos valores los toma del fichero myDefines.h.
2. **fncGetSts()** esta función devuelve el estado en el que se encuentra el relé. Si está apagado devolverá un cero, y si se encuentra encendido un uno.
3. **fncSetSts(unsigned short bStatusOnOff)** activa o desactiva el relé y el led asociado según el valor *bStatusOnOff* . Si recibe uno se conecta y en caso contrario se desconecta.

4.8.5.4. Programación relacionada con el reloj

Las funciones del reloj solo se utilizan dentro de *fncUpdateStatus()*.

El reloj que se está utilizando cuenta con una librería propia compatible con Arduino, la librería DS1307. Esta librería permite acceder a los datos del reloj tanto para leerlos como para modificarlos en caso de que el reloj no posea la fecha u hora correcta. Sin embargo esta librería no viene instalada por defecto en el IDE de Arduino por lo que es necesario hacerlo manualmente.

Guía para la instalación de la librería DS1307

Paso 1 Descarga : Esta librería es sencilla de encontrar en Internet aunque se recomienda descargarla de la página del fabricante PJRC.

`http://www.pjrc.com/teensy/td_ibs_DS1307RTC.html`

Paso 2 Crear Carpeta en directorio Arduino: Trás realizar la descarga se debe crear una carpeta con el nombre DS1307 en :

```
C:\<Program Files>\Arduino\libraries
```

Paso 3 Guardar librería: Finalmente en el directorio que acabamos de crear se debe copiar los dos archivos que se han descargado. Es necesario reiniciar el IDE de Arduino para que los cambios entren en funcionamiento.

Funciones

Una vez realizada la instalación podemos disponer de todas las funciones de esta librería. En concreto en este proyecto se están utilizando:

1. **clock.begin():** Inicia las comunicaciones con el reloj .
2. **clock.getTime():** Actualiza en los atributos del reloj los datos referentes a fecha y hora. Es necesario aclarar que esta función no retorna datos, sino que únicamente actualiza los parámetros: *clock.year*, *clock.month*, *clock.day*, *clock.hour*, *clock.minutes* y *clock.seconds*, por lo que para conocer la hora debemos actualizar, y después solicitar los valores de estos parámetros
3. **clock.fillByYMD(int Year, int month, int day):** Esta función permite al usuario preparar los valores relativos a la fecha, solo los prepara no los establece. Así, se puede configurar los parámetros del reloj al iniciarse por primera vez, o si los parametros no son correctos para el lugar de residencia del usuario. Esta función recibe como parámetros tres datos de tipo Int referentes a año, mes y día
4. **clock.fillByHMS(int hour, int minute, int second):** Esta función es muy similar a la anterior, pero destinada a preparar en el reloj los atributos relacionados con la hora, es decir hora, minutos y segundos.
5. **clock.setTime():** Las dos funciones anteriores solo preparan los atributos, y esta es la encargada de establecer realmente los cambios en el reloj.

Atributos

La clase DS1307 cuenta con una gran variedad de atributos, sin embargo en este código solo se van a utilizar los siguientes:

1. **Year** :Año
2. **Month**: Mes
3. **DayOfMonth**:Día del mes
4. **Hour**: Hora
5. **Minute**: Minuto
6. **Second**: Segundo

4.8.5.5. Función process()

La función *process()* recibe los comandos enviados por la página web, los procesa, y ejecuta los procesos pertinentes. Así su función es satisfacer las peticiones de los usuarios recogidas desde el entorno web.

Cuando un usuario realiza una petición desde la página, esta es recibida por el microprocesador de linux en forma de URL.

En caso de que la URL tenga formato: `http://<IPdeArduino>/Arduino` el microcontrolador linux manda a través del *Bridge* al ATmega32U4 la cadena de caracteres a partir de la palabra Arduino.

Por ejemplo si un usuario solicitara cambio de hora, se mandaría al procesador de linux la cadena correspondiente a la URL:

`http:\<IPArduino>/Arduino/SETTIME/<aaaa>/<mm>/<dd>/<hh>/<mi>`

Al tratarse de una cadena con formato: dirección seguido de la palabra Arduino, el procesador linux enviaría automáticamente a través del *Bridge* la siguiente cadena:

`SETTIME/<aaaa>/<mm>/<dd>/<hh>/<mi>`

Una vez recibida esta cadena en el ATmega32U4 la función *Process()* debe procesarla y ejecutar las ordenes. Para ello se descompone la cadena tomando como separador la barra. El primer parámetro es el tipo de orden , en el ejemplo *SETTIME*, el resto de parámetros se interpretan en función del primero, es decir, en función de la orden recibida.

Ejemplo del código encargado de la interpretación de una orden recibida desde la web:

```
void Process(YunClient client)
{
  String commandFull (client.readStringUntil('\r'));
  int iPos =  commandFull.indexOf('/') ;
  String command = commandFull.substring(0, iPos);

  if (command == CMCSETTIM)
  {
    //Descomponer los parametros separados por /
    ptr = strtok( aParameter, " / " );
    //Fecha
    ptr = strtok( NULL, " / " );
    g_iA = atoi(ptr); //
    ptr = strtok( NULL, " / " );
    g_iB = atoi(ptr); // Mes
    ptr = strtok( NULL, " / " );
    g_iC = atoi(ptr); // Día
    clock.fillByYMD(g_iA, g_iB, g_iC);
    clock.setTime();
    // Hora
    ptr = strtok( NULL, " / " );
    g_iA = atoi(ptr); // Hora
    ptr = strtok( NULL, " / " );
    g_iB = atoi(ptr); // Minutos
    clock.fillByHMS(g_iA, g_iB, 0);
    clock.setTime();
    client.println( 1); // Devuelve el código al cliente
    return;
  }
```

Nota: En un primer momento las ordenes se enviaban con una cadena de texto significativa como *settime*. Sin embargo por problemas de espacio esto se modificó para reducir el consumo de memoria del procesador, pasando a utilizar un número entero identificador de cada orden que queda definido en el fichero *myDefines.h* como *SETTIME*.

4.8.5.6. Función fncConfigSave()

En caso de desconexión, el smartplug debe ser capaz de recuperar la última configuración establecida por el usuario. Así ante incidencias como caídas de tensión o desconexiones accidentales, el smartplug es capaz de continuar en funcionamiento sin necesidad de reconfigurar todas sus opciones.

Esto se ha solucionado grabando la configuración y el estado en la microSD con la función *fncConfigSave()*, esta función es llamada desde la función *Process()* después de ejecutar cualquier petición web que implique cambios en la configuración. La función *process()* antes de llamar a *fncConfigSave()* llama a la función *fncUpdateStatus()* que actualiza los valores en los buffer.

```
void  fncConfigSave()
{
  if (FileSystem.exists(g_cFileName))
  {
    FileSystem.remove(g_cFileName);
  }
  File FileS = FileSystem.open(g_cFileName, FILE_WRITE);
  FileS.print(g_BufferStd);
  FileS.flush();
  FileS.print(g_BufferMod);
  FileS.flush();
  FileS.close();
}
```

4.8.5.7. Función fncConfigBegin()

Esta función se encarga de establecer la configuración inicial al arrancar el Arduino a partir del fichero de configuración denominado: `/mnt/SDA/arduino/smartplug<nn>.cfg/` tal como se ve en la figura 4.11.

Previniendo el posible cambio de la tarjeta microSD en cuyo caso no existiría fichero de configuración, el proceso lo crea con valores por defecto.

Ya que esta función solo se debe ejecutar cuando arranca el Arduino se realiza dentro del bloque *Setup()*

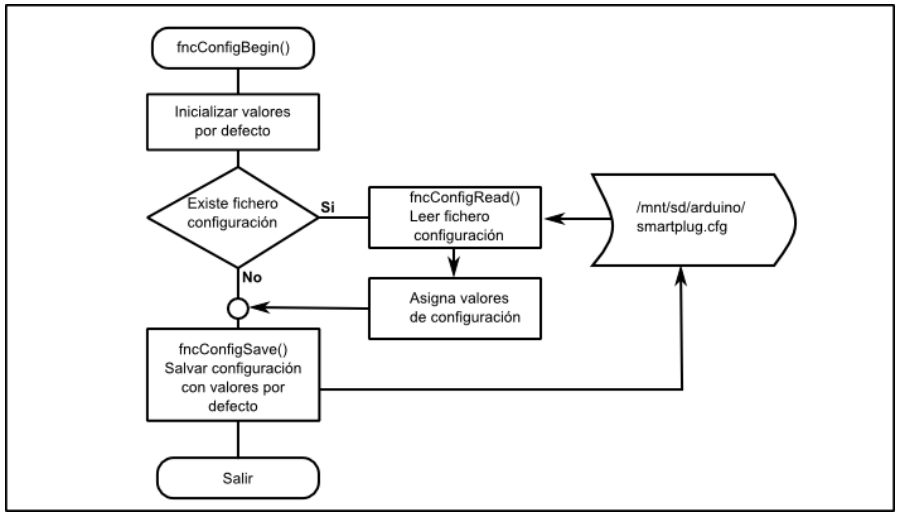


Figura 4.11: Diagrama de flujo de la función fncConfigBegin()

4.8.5.8. Función fncRead();

Lee la configuración guardada en el fichero: `/mnt/SDA/arduino/smartplug<nn>.cfg` almacenándola en un array que contiene la fecha y hora del momento de salvado, el estado y configuración de los relés y el estado del zumbador. Además tiene un espacio adicional de 25 caracteres reservado para los módulos especializados.

Esta función solo es llamada desde el *fncConfigBegin()*, donde se utiliza para leer la última configuración escogida por el usuario cuando arranca el Arduino. Consultar figura 4.11.

4.8.5.9. Función fncUpdateStatus();

La finalidad de la función *fncUpdateStatus()* es leer el reloj, los sensores (en módulos especializados) conectados al Arduino, y procesar esta información de acuerdo a la programación para activar o desactivar los relés según convenga.

En el módulo estándar esta función solo permite con la información recogida programar los relés. En los módulos específicos se amplían las funcionalidades pudiendo no solo programar los relés en más modos, sino también ejecutar acciones como enviar emails, activar motores ...

Por ejemplo el Smartplug01 es un módulo de control ambiental que ha sido desarrollado para que utilizando un sensor de temperatura y humedad, se activen: un calefactor, un ventilador u otros dispositivos.

Otro ejemplo es el segundo módulo desarrollado, el Smartplug02, un módulo de alarma que añadiendo un sensor de gas y presencia permite de encender los distintos dispositivos al detectar intrusos o gas.

Esta función es ejecutada desde el bucle *loop()* y desde la función *fncFillStatus()*.

4.8.5.10. Función fncFillStatus()

Se encarga de rellenar los buffers: *g_BufferStd, g_BufferMoD, y g_BufferWeb* a partir de las variables globales que contienen la configuración y el estado de los dispositivos.

Esta función se ejecuta desde la función *Process()* con el fin de enviar la información de estado y configuración a la página web. Además se ejecuta desde la función *fncConfigBegin()* para actualizar el estado tras arrancar el Smartplug.

Dentro de *fncFillStatus()* primero se llama a *fncUpdateStatus()* para asegurarse que los valores a rellenar en el buffer se corresponden a la última situación de sensores y relés.

Nota: Realmente solo sería necesario el *g_BufferWeb*, pero se han tenido que declarar dos buffers adicionales (*g_BufferStd*, *g_BufferMoD*) que en realidad son una división del *g_BufferWeb* debido a problemas de grabación de cadenas largas en la tarjeta microSD. Estos problemas ya están reportados a Arduino y están pendientes de solución.

En la tabla que se encuentra a continuación se explica el significado de cada uno de los parametros que contiene *g_BufferWeb*.

Nº	Dispositivo	Variable	Lng	Ini	Fin	Rango	Nota
1	Reloj	Año	4	0	3	[2000-9999]	
2		Mes	2	4	5	[1-12]	
3		Dia	2	6	7	[1-31]	
4		Hora	2	8	8	[0-11]	
5		Minuto	2	10	11	[0-59]	
6		Segundo	2	12	13	[0-59]	
7	Zumbador	BuzerBeeping	1	14	14	[0-1]	0=Silencio, 1=Sonando
8	Relé 1	a Status	1	15	16	[0-1]	Relé conectado o desconectado
9		b PrgMode	1	17	17	[0-5]	Leer nota 1
10		c SlotStar	4	18	21	[0-719]	Inicio del intervalo expresado en minuto del día
11		d SlotEnd	4	22	25	[0-719]	Fin del intervalo expresado en minuto del día
12		e SlotMode	1	26	26	[0-1]	0=conectar, 1=desconectar. Leer nota 2
13	Relé 2	a Status	1	27	28		Se repite la interpretación del relé 1
14		b PrgMode	1	29	29		
15		c SlotStar	4	30	33		
16		d SlotEnd	4	34	37		
17		e SlotMode	1	38	38		
18	Relé 3	a Status	1	39	39		Se repite la interpretación del relé 1
19		b PrgMode	1	40	40		
20		c SlotStar	4	44	44		
21		d SlotEnd	4	45	48		
22		e SlotMode	1	49	49		
23	Relé 4	a Status	1	50	50		Se repite la interpretación del relé 1
24		b PrgMode	1	51	54		
25		c SlotStar	4	55	58		
26		d SlotEnd	4	59	62		
27		e SlotMode	1	63	63		
28		EndMark	1	64	64		Leer nota 4
29	Leer nota 3	Libre	25	65	89		

Tabla 4.3: Significado de parámetros del g_BufferWeb

Nota 1: Los modos de programación de los relés son: 0 (Manual OFF), 1 (Manual ON) y 2 (Automático condicionado a intervalo horario). Para más información en 4.8.5.3

Nota 2: Este valor solo afecta cuando el relé se haya configurado en modo de programación 2. El valor 1 indica que el relé estará conectado dentro del intervalo y desconectado fuera, mientras que el valor 0 indica se conectará fuera del intervalo y desconectará dentro

Nota 3: Espacio reservado para módulos especializados.

Nota 4: El único valor permitido es # Este valor indica el final del buffer del módulo estándar. Los campos precedentes a él no pueden ser cambiados en los módulos especializados.

A continuación se muestra un ejemplo de un BufferWeb. Las filas A1 contienen la descripción o la letra que indica el contenido según la tabla anterior. Las filas A2 contienen las posiciones, y la fila A3 contiene un ejemplo de buffer correspondiente al Smartplug01 que es un módulo de control ambiental.

	BUFFER MODULO ESTANDARD																								(*) ESPACIO RESERVADO PARA MODULO ESPECIFICO									
A1	Fecha y Hora		RELE 1					RELE 2					RELE 3					RELE 4						#	Actual		Pogramacion					D	Libre
	AAAAMMDDHHMMSS	Z	a	b	c	d	e	a	b	c	d		a	b	c	d	e	a	b	c	d	e		T	H	TD	TF	TV	HD	HF	HV		
A2	00000000001111	1	1	1	1112	2222	2	2	2	2233	3333	3	3	3	3444	4444	4	4	4	5555	5555	5	5	6666	6666	66	67	77	777	777	788	8	88888889999999999
	01234567890123	4	5	6	7890	1234	5	6	7	8901	2345	6	7	8	9012	3456	7	8	9	0123	4567	8	9	0012	3456	78	90	12	345	678	901	2	345678901234567890
A3	20140812105636	0	0	0	0480	1200	1	0	0	0480	1200	1	0	0	0480	1200		0	0	0480	1200	1	#	9999	9999	30	15	02	040	060	010	7	

Figura 4.12: Ejemplo de g_BufferWeb

4.8.6. Programación relacionada con AR9331

El microcontrolador AR9331 es el encargado de ejecutar los scripts en Python, leer y escribir en la tarjeta microSD, y actuar como servidor Web.

4.8.6.1. Python Script

Algunos modelos del smartplug cuentan con una aplicación para notificar alertas por email. Esto se realiza mediante un script de python almacenado en la microSD, y modificando la configuración de Arduino.

Guía de configuración de Arduino para mandar emails

Es importante recordar que estas modificaciones solo se incluyen en aquellos modelos que incluyen una funcionalidad para mandar emails.

Paso 1 Instalación de librerías: Utilizando el programa Putty (consular apéndice A) introducir las siguientes sentencias:

```
opkg update
opkg find python-openssl
opkg install python-openssl
```

Paso 2 Configuración en Luci: Utilizando el entorno Luci (consultar apéndice D) entrar en el menú System/startup. Una vez ahí situarse en el textbox de la parte inferior titulado Local Startup y modificarlo de la siguiente forma:

```
# Put your custom commands here that should
# be executed once the system init finished.
# By default this file does nothing.
wifi-live-or-reset
cd /mnt/sda1/Arduino/www/smartplug02/py
python sendmail.py
exit 0
```

Esto hace que al iniciarse el Arduino arraque el script de python *sendmail.py* encargado del envío de correo. Este script se explicará detalladamente más adelante.

Paso 3 Configuración del servidor: Finalmente hay que configurar el servidor web para que sea capaz de interpretar scripts de python. Para ello utilizando el programa WinSCP (consultar apéndice B) debemos editar el fichero */etc/config/uhttpd*y añadir esta linea.

```
list interpreter ".py=/usr/bin/python"
```

Una vez ejecutado este cambio hay que reiniciar el Arduino.

Cabe mencionar que para subir los scripts de python al Arduino es necesario guardarlos en la misma carpeta en la que se encuentre el código de programación del ATmega32U4, concretamente en la carpeta */mnt/SD/Arduino/www/smartplug(nn)/PY*

4.8.6.2. Lenguajes empleados en las páginas web

Las páginas web se encuentran diseñadas usando el estándar HTML5, Javascript y hojas de estilos siguiendo el estándar CSS3. Esto ha permitido desarrollar páginas cuyo diseño sea adaptable tanto a teléfonos móviles como a tablets y ordenadores, es decir son adaptables a cualquier dispositivo independientemente del tipo de pantalla y del sistema operativo siempre y cuando se tenga un navegador compatible con HTML5 como: Safari 5.1, Chrome 18 , Internet explorer 9, Firefox 12, Opera 11.6 y versiones sucesivas

La ventaja de este diseño es que se puede controlar el Smartplug desde cualquier dispositivo sin tener que cargar ningún software adicional.

Se ha tratado de hacer que el cliente web ejecute la mayor parte de los procesos a realizar para evitar sobrecargar el Arduino, para ello se ha empleado el Javascript combinado con la librería Jquery y los propios scripts desarrollados.

Los mensajes entre las páginas y el Arduino se han desarrollado utilizando la tecnología Ajax y el estándar Json.

4.8.6.3. Estructura de directorios de la web

Arduino necesita una estructura de carpetas específica para hacer funcionar las páginas web. Esta sigue el siguiente patrón:

```
/mnt/SDA1/Arduino/www/<Nombre del programa>/<Js,Css,Img>
```

Es decir, necesita tener dentro de la microSD una carpeta llamada *arduino* y dentro de esta una llamada *www* y después una con el nombre del programa, en esta carpeta se guardan las páginas HTML5, a partir de ahí la estructura es libre.

En este proyecto se ha optado por hacer una carpeta para cada tipo de lenguaje de las páginas web para facilitar el desarrollo . Así, existen tres carpetas :

1. **css:** Contiene la hoja de estilos e imágenes relacionadas con ella.
2. **img:** Contiene imágenes generadas por el programa.
3. **js:** Contiene los archivos Javascript.

Nota: en caso de existir scripts de python también hay una carpeta *Py*.

Durante el desarrollo es muy importante almacenar todos los archivos relacionados con la página web en una subcarpeta del proyecto denominada *www*, ya que así con en interfaz IDE del Arduino subirán al mismo tiempo que el programa.

4.8.6.4. Circulación de la información entre cliente web y el Arduino

1 El cliente pide la página http://../smartplug01/index.html

2 Recibe la página estática y los ficheros auxiliares (css, js).

3 Al completar la carga de la página estática se deben rellenar los valores dinámicos. Para ello se llama a la función *fncOnload()* que solicita los datos actuales del sistema llamando a la URL: */arduino/0/*.

Esta petición al contener la cadena */arduino/* es redirigida por el linux al ATmega32u4, que la procesa y envía de vuelta en un buffer que contiene todos los datos.

4 Una vez se ha recibido el buffer, se llama a la función *fncfill()* encargada de rellenar esta información en la página web.

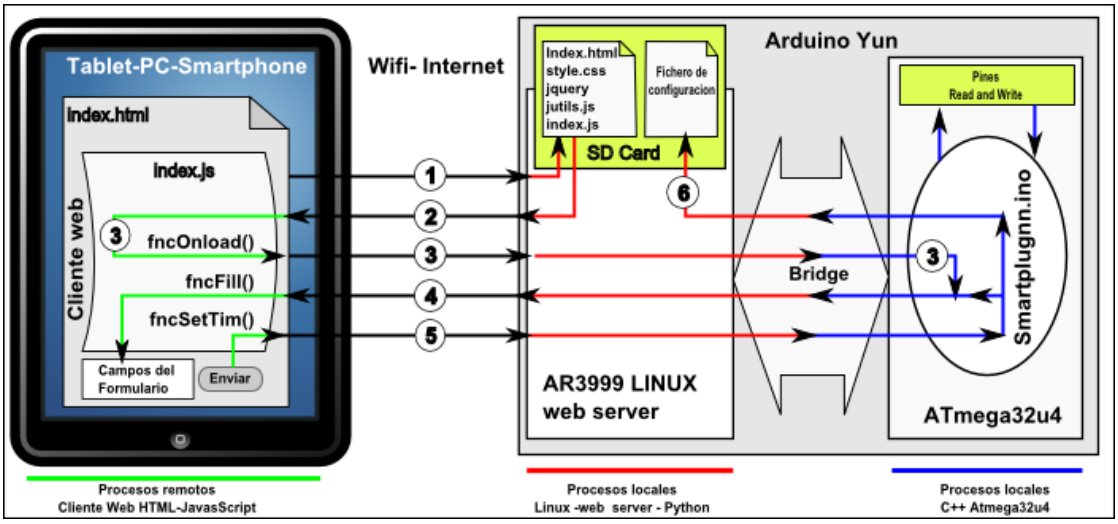


Figura 4.13: Circulación de la información entre cliente y servidor.

5 Si el usuario solicita un cambio de configuración se ejecutan las funciones AJAX tipo *fncsetXXX()*.

Por ejemplo si se pidiera un cambio de hora, se llamaría a la función *fncSetTim()*, que mediante una petición URL solicitaría que se actualizara la fecha y hora. Esta petición llegaría a través del *Bridge* al ATmega32u4 que cambiaría a la hora, guardaría la configuración en la tarjeta SD (6) y devolvería el resultado.

6 Cuando se realizan cambios en la configuración estos se guardan en la microSD. De esta forma si el equipo se reinicia puede recuperar los parámetros programados.

Nota: Los comandos que y procesos se explican con detalle en sus correspondientes apartados.

4.8.6.5. Páginas web incluidas en el módulo general

Dentro del módulo general se encuentran las páginas relacionadas con el estado y la configuración del reloj y los relés. Estas son:

1. **Index.html:** Muestra el estado del dispositivo. A partir de ahora se denominará página de estado.
2. **Config.html:** Permite configurar el reloj y en caso de un módulo específico se añaden otras opciones. A partir de ahora se denominará página de configuración.
3. **Configplugs.html:** Permite configurar todas las opciones relacionadas con los enchufes, como son los modos de programación o los intervalos. A partir de ahora se denominará página de configuración de enchufes.

4. **Guide.html:** Es una guía básica de uso destinada al usuario final. A partir de ahora se denominará página de guía al usuario.

Para acceder a la página de estado (index.html) debemos introducir en cualquier navegador la siguiente dirección:

```
http://<Dirección Ip arduino>/SD/<nombre del programa>
```

A continuación se va a proceder a explicar cada una de las páginas de módulo junto a sus funciones más relevantes.

Página de Estado

Es la primera página que aparece al introducir la dirección anterior. Está dividida en secciones que se visualizan enmarcadas en recuadros. La primera sección muestra valores de fecha, hora y posibles sensores, las siguientes muestran el estado y modo de programación de cada uno de los enchufes. Además en la parte superior aparece un menú para dirigirse al resto de páginas.

Cabe mencionar que esta página es puramente informativa, por lo que no permite modificar ningún parámetro.

Figura 4.14: Página de Estado del módulo estándar

Para el funcionamiento de esta página se han empleado las siguientes funciones Javascript:

1. **fncLoad():** Se dispara solo una vez cuando finaliza la carga de la página y llama a la función *fncgetRefresh()*.

2. **fncgetRefresh():** Se llama cada vez que necesitamos visualizar los datos actualizados desde el Arduino, evitando así tener que recargar toda la página. Esto lo realiza solicitando mediante Ajax al Arduino la URL `/arduino/0` cuya respuesta es el envío del *g_BufferWeb* con toda programación y estado del dispositivo. Una vez recibida la respuesta llama a la función *fncFillStatus()*.

3. **fncFill():** Esta función toma el buffer de datos enviado desde el procesador ATmega32U4, lo separa y distribuye entre los distintos campos de la página web.

Cabe mencionar que en las primeras versiones de este software los datos se enviaban individualmente según las distintas peticiones del usuario desde la página web. Sin embargo, esto consumía una gran cantidad de memoria del procesador ATmega32U4 cuyo espacio es muy reducido. Finalmente, se optó por enviar todos los datos en una misma cadena y que las páginas web, utilizando Javascript usaran los datos que necesitaran.

Página de configuración

Esta página en el módulo general permite configurar la hora y fecha del reloj (figura 4.15) además al abrirla muestra en unos combo los valores actuales de configuración. En los módulos especializados en esta página se añadirán las configuraciones relativas a los sensores.

El usuario puede modificar los valores correspondientes a año, mes, día, hora y minuto, y seguidamente pulsar el botón enviar. Durante la carga de la página aparece una imagen indicando al usuario que debe esperar e impidiendo peticiones reiterativas al servidor.

Figura 4.15: Página de configuración

Para configurar los parametros del reloj se han elaborado las siguientes funciones en Javascript contenidas en los ficheros: *config.js* y *utils.js*:

1. **fncLoad():** Se dispara solo una vez cuando finaliza la carga de la página y llama a la función *fncgetRefresh()*.

2. **fncGetRefresh():** Primero comprueba si el servidor esta atendiendo a una petición anterior mediante la función *fncLoadingStatus():* y en caso de estar libre solicita mediante Ajax la petición: *\arduino\0* y el servidor responde enviando el *g_BufferWeb*, a continuación el Javascript llama a la función *fncFillCombos()*.

3. **fncFillCombos():** Toma los datos que necesita del *g_BufferWeb* y los distribuye en los distintos combos.

4. **fncMessageShow(sMessage)**: Se utiliza para poner mensajes informativos en la página web.

5. **fncSetTim()**: Solicita al servidor el cambio de fecha y hora grabada en el reloj de tiempo real. Esto lo realiza enviando una solicitud Ajax con el siguiente formato *\arduino\1\año\mes\día\hora\minuto*. Una vez realizado el cambio, el servidor le responde con el *g_BufferWeb* y se recarga la página con la información actualizada usando las funciones pertinentes.

Página de configuración de enchufes

Permite modificar la configuración de los enchufes. Además desde esta página también se puede leer la hora del reloj para facilitar al usuario la programación de intervalos.

Figura 4.16: Página de configuración de enchufes

Sus funciones son :

1. **fncLoad():** Se ejecuta al acabar de cargar de la página y llama a la función *fncgetRefresh()*.
2. **fncGetRefresh():** Al igual que en la página de configuración comprueba si el servidor está atendiendo a una petición anterior y en caso de estar libre solicita mediante Ajax la petición: *\arduino\0* y con la respuesta llama a la función *fncFillCombos()* .
3. **fncFillCombos():** Rellena en los combos de la página los valores de configuración actuales.
4. **fncSetRelay(IdPlug):** Esta función se ejecuta al pulsar el botón enviar en la página de configuración de enchufes. Recibe el número del enchufe y el modo de programación que se desea configurar y envía mediante Ajax la petición:

   \arduino\5\<idPlug>\<prgmode>\<slotstart>\<slotend>\<slotinonoff>

   Donde :

   Arduino: Le indica al servidor que debe mandarlo utilizando el *Bridge* al ATmega32U4.

   5: Es el tipo de orden, 5 corresponde a programación de relés

   IdPlug: Es una variable que contiene el número de enchufe.

   Prgmode: Es una variable que indica el modo de programación.

   Slotstart: Es una variable que contiene el minuto del día en el que se inicia el intervalo.

   Slotend: Es una variable que contiene el minuto del día en el que se acaba el intervalo.

   Slotinonoff: Indica si el usuario desea que se conecte dentro o fuera del intervalo.

Este comando se manda a través del *Bridge* al ATmega32U4 que actualiza el modo de configuración, y devuelve el nuevo *g_BufferWeb* al microprocesador linux.

Página de guía al usuario

Su función es contener una sencilla guía para el usuario final. En ella se describen las funcionalidades comunes como la guía para la conexión a la WIFI y en módulos especializados las funcionalidades características.

Funciones Javascript comunes a todas las páginas

Existen un conjunto de funciones comunes a varias páginas, por ello se ha creado un archivo Javascript común a todas denominado *utils.js*.

Las funciones comunes son las siguientes:

1. **fncLoadingStatus():** Permite mediante un campo oculto evitar llamadas sucesivas al servidor mientras esta procesando una petición anterior. Devuelve uno si se esta procesando y en caso contrario cero.
2. **fncLoadingStart():** Muestra la barra de cargando (figura 4.17) y pone a uno el campo oculto.
3. **fncLoadingEnd():** Oculta la barra de cargando (figura 4.17) y pone a cero el campo oculto.

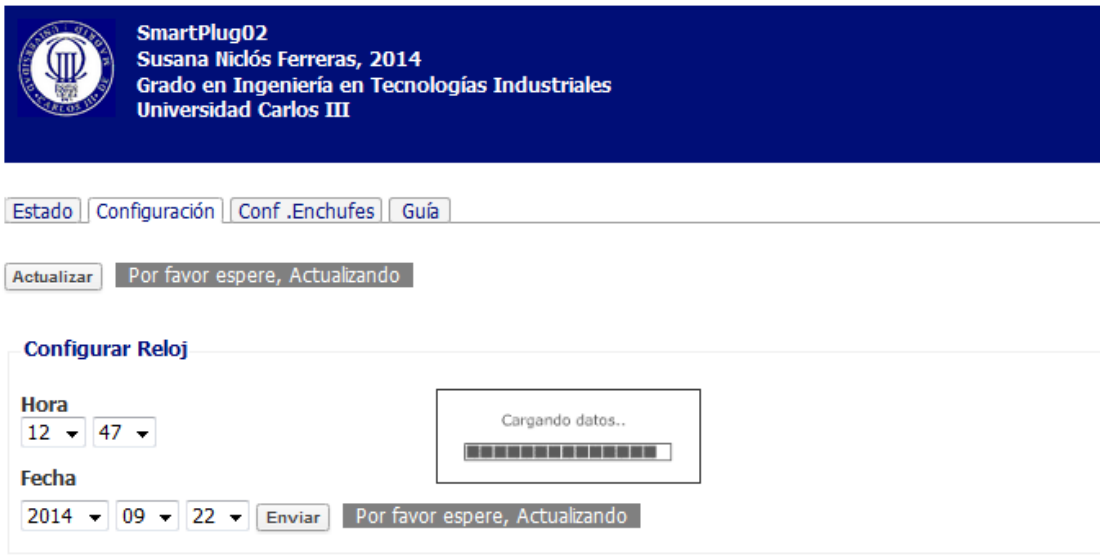


Figura 4.17: Barra de actualización de las páginas web

Nota: Las tres funciones manejan el campo oculto en la página web denominado *isloading* cuyo valor cero representa que la página esta libre para realizar nuevas peticiones y uno para indicar que la página esta ocupada.

Capítulo 5

Guía para crear nuevas aplicaciones 1: Módulo de control ambiental, Smartplug01

Este módulo tiene como objetivo mostrar con un ejemplo la versatilidad del módulo estándar, y servir de guía para el desarrollo de nuevas aplicaciones especificas. Añadiéndole simplemente un sensor de temperatura y humedad, como el DHT22 y modificando la programación, el kit se transforma en un sistema capaz de controlar la temperatura, humedad, iluminación y ventilación de un recinto.

Figura 5.1: Módulo de control ambiental

En la siguiente tabla se muestra una comparativa de las funcionalidades:

Funcionalidades añadidas o reutilizadas.		
Funcionalidad	Módulo estándar	Módulo Ambiental
Servidor Web	✓	✓
Configuración programación mediante Web	✓	✓(Ampliada)
Acceso Wifi Internet	✓	✓
4 Relés que controla 4 enchufes	✓	✓
Modo programación manual Manual off	✓	✓
Modo programación manual Manual on	✓	✓
Modo programación por intervalos de tiempo	✓	✓(Iluminación)
Modo programación en función de temperatura y dos intervalos de tiempo		✓(Calefactor)
Modo programación en función de humedad y dos intervalos de tiempo		✓(Humificador)
Modo programación en función de temperatura, humedad y dos intervalos de tiempo		✓(Ventilación)

Tabla 5.1: Funcionalidades del modulo ambiental.

5.1. Hardware añadido y conexiones

Este módulo cuenta con el sensor de temperatura y humedad DHT22, que opera de 3.3 a 6 V DC con un consumo de 1.5mA max. (50uA en stand-by). Este sensor es capaz de medir un rango de temperaturas de -40 a +125°C con una precisión de ±0.2 °C y un rango de humedad de 0 a 100 %, con una precisión de ±2 (máx ±5 %) [24].

5.1.1. Conexiones

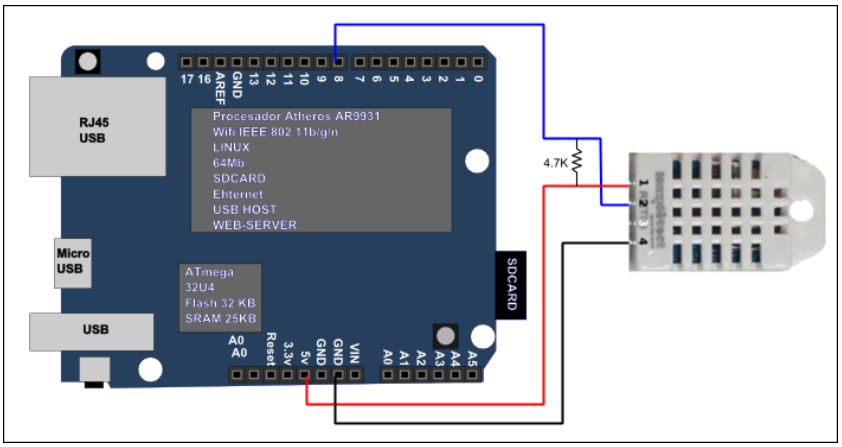


Figura 5.2: Esquema conexiones DHT22 ATmega32U4

El sensor de temperatura tiene cuatro pines que se conectan de la siguiente forma:

- **Pin 1 Vcc:** Conectado a la salida de 5 V de la placa Arduino.
- **Pin 2 Data:** Conectado al pin digital 8, temperatura y humedad comparten el pin.
- **Pin 3:** Sin Uso.
- **Pin 4:** Conectado a GND de Arduino.

  Nota: Hay que colocar una resistencia Pull-Down 4.7 kΩ entre el pin de alimentación (1) y el de datos (2).

5.2. Software, creación del nuevo proyecto

Las modificaciones de la programación a partir de software desarrollado para el módulo estándar incluyen la modificación tanto de la parte desarrollada con el IDE de Arduino realizada en C++ destinada al ATmega32U4 como la parte desarrollada para la web realizada en HTML y Javascript que utiliza el procesador AR9331(Linino-Linux).

5.2.1. Creación de la base para el nuevo proyecto. Smartplug01

Tomando como plantilla del proyecto el software desarrollado para el módulo estándar denominado Smartplug00, se crea el nuevo proyecto Smartplug01, (Control ambiental). Para ello se siguen los siguientes pasos:

1. Crear una carpeta con el nombre Smartplug01, que contendrá el nuevo proyecto.
2. Copiar el contenido de la carpeta Smartplug00 en la carpeta Smartplug01
3. Situarse en la carpeta Smartplug01, redenominar el archivo smartplug**00**.ino a smartplug**01**.ino

Nota: TODO, por esta palabra comienzan numerosas lineas de comentario del código del módulo estándar, tanto de C++ y HTML, como de Javascript. Esta palabra aparece seguida del comentario que indica que tipo de código se puede añadir en esa posición. Así es posible mantener una estructura estandarizada en todos los módulos específicos.

5.3. Modificaciones de software realizadas para el ATmega32U4

Es necesaria la librería dht(2014, Rob Tillaart. VERSIÓN: 0.1.13), esta versión es mucho más estable y utiliza más eficientemente la memoria, al permitir con una sola instancia leer múltiples sensores conectados a diferentes pins. Es descargable desde `http://arduino.cc/playground/Main/DHTLib/`

Para el correcto funcionamiento en el IDE de Arduino los ficheros dht.h y dht.cpp deben ser copiados a la carpeta: `c://ProgramFiles/Arduino/libraries/dht`. Para más información sobre la instalación de librerías consultar 4.8.5.4.

5.3.1. Proyecto Smartplug01 modificaciones fichero MyDefines.h

Aquí se agregan las constantes tipo *defines* que se usan para declarar pines, modos de programación y comandos web.

TODO 01 Cambiar el nombre del fichero de configuración.

```
#define CONFIGFILENAME "/mnt/microSD/smartplug01.cfg"
```

TODO 02 Agregar a partir de aquí las constantes para pines específicos.

```
#define PIN_DHT2 8
```

TODO 03 Agregar a partir de aquí las constantes que se usan para modos de programación de los relés

```
#define PRGAUTOCLOCK     2 // Se reutiliza para iluminación
#define PRGAUTOTEMP      3 // Calefacción - Temperatura
#define PRGAUTOHUME      4 // Humificador - Humedad
#define PRGAUTOTEMPHUME 5 // Ventilación - Temp. y Hum
```

TODO 04 Agregar a partir de aquí valores por defecto.

```
#define DEFAULTTEMSLOTIN 300
// Temperatura dentro del intervalo programado: 30ºC
#define DEFAULTTEMSLOTOUT 150
// Temperatura fuera del intervalo programado: 15ºC
#define  DEFAULTTEMSLOTVAR 20
// Variación admitida de la temperatura programada: 2ºC
#define DEFAULTHUMSLOTIN 400
// Tanto por ciento de humedad relativa dentro del intervalo: 40
```

```
#define DEFAULTHUMSLOTOUT 600
// Tanto por ciento de humedad relativa fuera del intervalo: 60
#define DEFAULTHUMSLOTVAR 100
// Tanto por ciento de variación de humedad admitida: 10
```

Nota: Estos valores se toman si no existe una configuración modificada por el usuario, o si se cambia la tarjeta SD.

TODO 05 Agregar a partir de aquí los identificadores de comandos web.

```
#define CMDSETTEM "3"
// Comando recibido desde web indicando cambios en configuración
// de la programación de temperatura.
// Ejemplo: Cambiar temperatura (3) a 30ºC dentro del intervalo,
// 18ºC fuera y un grado de variación, se hace recibiendo desde
// la web la petición:
// URL: arduino/3/300/180/10
#define CMDSETHUM "4"
//Comando recibido desde web indicando cambios en la programación
// de humedad.
// Ejemplo: URL: arduino/4/600/800/100
```

TODO 06 Agregar a partir de aquí los otros valores.

```
#define CALIBRATIONTIMESS 10
// Espera para que el sensor se calibre
```

5.3.2. Proyecto Smartplug01 modificaciones fichero smartplug01.ino.

TODO 01 Includes, librerías ficheros .h adicionales.

```
#include < dht.h> //Librería del sensor de temperatura.
```

TODO 02 Variables globales del módulo específico. Se aconseja utilizar el prefijo **gm_** (global módule) para evitar posibles redefiniciones de variable y facilitar la lectura del código.

```
byte    gm_bDht22Sts;
\\Código devuelto tras la lectura del sensor dht22
int  gm_bPrgTemSlotIn =  DEFAULTTEMSLOTIN;
// Temperatura dentro del intervalo horario
int  gm_bPrgTemSlotOut = DEFAULTTEMSLOTOUT;
// Temperatura fuera del intervalo horario
int  gm_bPrgTemSlotVar = DEFAULTTEMSLOTVAR;
// Variación tolerada, evita on/off demasiado rápido
```

```
int  gm_bPrgHumSlotIn = DEFAULTHUMSLOTIN;
// Humedad programada dentro del intervalo horario décimas
int  gm_bPrgHumSlotOut = DEFAULTHUMSLOTOUT;
// Humedad programada fuera del intervalo horario
int  gm_bPrgHumSlotVar = DEFAULTHUMSLOTVAR;
// Variación tolerada
int gm_bActHum = 0;
// Humedad actual leída del DHT22
int gm_bActTem = 0;
// Temperatura actual leída del DHT22
int gm_bActTemPrgMax = 0;
// Temperatura máxima establecida para el momento actual
int gm_bActTemPrgMin = 0;
// Temperatura mínima establecida para el momento actual
int gm_bActHumPrgMax = 0;
// Humedad máxima establecida para el momento actual
int gm_bActHumPrgMin = 0;
// Humedad mínima establecida para el momento actual
byte  gm_Flag_SwithcTem = 0;
// 1 = Conectar, 0 = Desconectar.
//(Dispositivo condicionado a temperatura, calefactor)
byte  gm_Flag_SwithcHum = 0;
// 1 = Conectar, 0 = Desconectar.
// (Dispositivo condicionado a humedad)
byte  gm_Flag_SwithcTemHum = 0;
// 1 = Conectar, 0 = Desconectar
// (Dispositivo condicionado a humedad y temperatura, ventilador)
```

TODO 03 Agregar al switch que controla los modos de programación estándar las funcionalidades adicionales y llamar a las nuevas funciones que las controlan.

Se agregan funciones para la lógica de control de temperatura, humedad y ventilación, que se programan aparte para más claridad, en las funciones *fnc_PRGAUTOTEMP(n)*, *fnc_PRGAUTOHUME(n)*, *fnc_PRGAUTOTEMPHUME(n)*, donde n es el identificador del relé sobre el que se esta actuando.

```
case PRGAUTOTEMP:
fnc_PRGAUTOTEMP(n);
g_bOnOff = gm_Flag_SwithcTem;
break;
case PRGAUTOHUME:
fnc_PRGAUTOHUME(n);
g_bOnOff = gm_Flag_SwithcHum;
break;
case PRGAUTOTEMPHUME:
fnc_PRGAUTOTEMPHUME(n);
g_bOnOff = gm_Flag_SwithcTemHum;
break;
```

fnc_PRGLimitsTempHum(n): Cálculo de límites para el intervalo actual.

Esta función tiene por objeto calcular los valores límites de temperatura y humedad para el instante actual, en función de la programación y de si se está dentro o fuera del intervalo programado para un relé determinado.

Lógica para calcular la temperatura y humedad máxima y mínima a aplicar al momento actual:

Si se está dentro del intervalo, la temperatura máxima es la programada para el intervalo sino, la máxima es la programada para fuera del intervalo. Una vez obtenida esta, la mínima igual a la máxima menos la variación tolerada. El mismo razonamiento se usa para la humedad.

```
void fnc_PRGLimitsTempHum(){
  if (g_Flag_SwithcInt == 1)
  {
    gm_bActTemPrgMax = gm_bPrgTemSlotIn ;
    gm_bActHumPrgMax = gm_bPrgHumSlotIn ;
  }
  else
  {
     gm_bActTemPrgMax = gm_bPrgTemSlotOut ;
     gm_bActHumPrgMax = gm_bPrgHumSlotOut ;
  };
gm_bActTemPrgMin = gm_bActTemPrgMax - gm_bPrgTemSlotVar;
gm_bActHumPrgMin = gm_bActHumPrgMax - gm_bPrgHumSlotVar;
if (gm_bActHumPrgMin < 0)gm_bActHumPrgMin = 0;
}
```

fnc_PRGAUTOTEMP(int n): Cálculo del switch de temperatura para el intervalo actual. Tiene por objeto calcular la variable *gm_Flag_SwithcTem* para el relé n, que valdrá 1 en el caso de que se deba conectar el relé, si este está condicionado a la temperatura. (Calefactor)

Lógica para encender o apagar el calefactor:

1 Si la temperatura es inferior al mínimo programado, conectar calefacción hasta que se alcance el máximo, *gm_Flag_SwithcTem=1.* (Fig. 5.3 Pto. 1).

2 Si la temperatura sobrepasa el máximo programado apagar calefacción,*gm_Flag_SwithcTem=0.* (Fig. 5.3 Pto. 2).

3 y 4 Los puntos de la figura 3 y 4 afectan a la ventilación, y se verán en el apartado correspondiente.

5 Si la temperatura desciende del máximo mantener apagada la calefacción, *gm_Flag_SwithcTem=0.* (Fig 5.3 Pto. 5).

6 Si la temperatura desciende del mínimo programado encender de nuevo la calefacción,*gm_Flag_SwithcTem=1.* (Fig 5.3 Pto. 6).

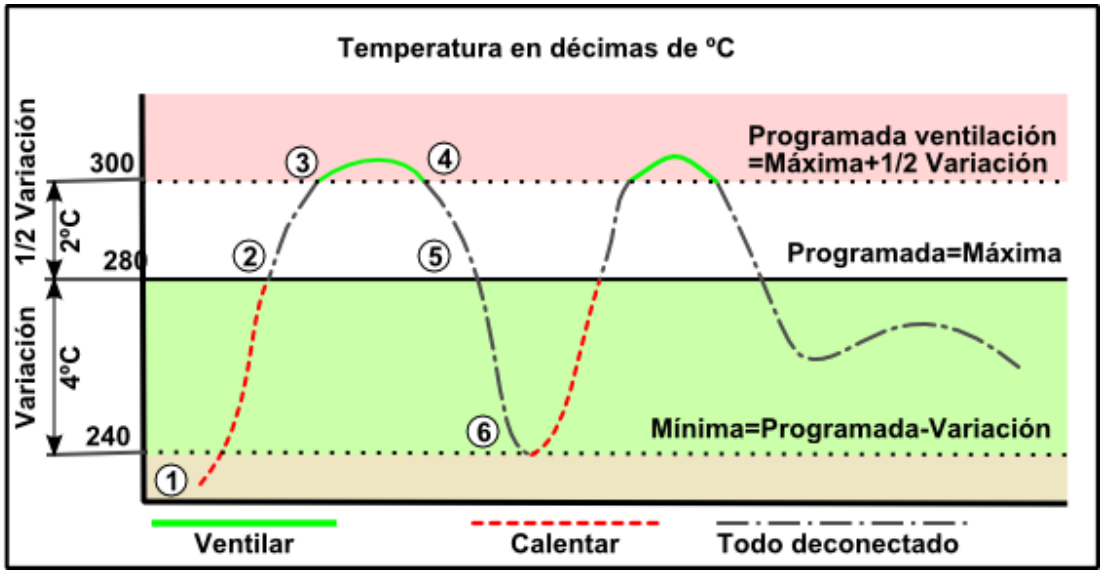


Figura 5.3: Gráfica de lógica de control de temperatura

```
void fnc\_PRGAUTOTEMP(int n)
{
fnc_CalcFlag_SwithcInt(n);
fnc_PRGLimitsTempHum();
if (gm_bActTem >= gm_bActTemPrgMax )
   {
      gm_Flag_SwithcTem = 0;
   }
   else
   {
            if (gm_bActTem <= gm_bActTemPrgMin )
            {
               gm_Flag_SwithcTem = 1;
            }
        }
}
```

fnc_PRGAUTOHUM(int n): Cálculo del switch de humedad. Tiene por objeto calcular *gm_Flag_SwithcHum* para el relé n, que valdrá 1 si se debe encender el relé siempre y cuando este este condicionado a la temperatura.

Lógica para encender o apagar el humificador

1 Si la humedad es inferior al mínimo programado, conectar humificador hasta que se alcance el máximo, *gm_Flag_SwithcHum=1*. (Fig. 5.4 Pto. 1).

2 Si la humedad sobrepasa el máximo programado apagar humificador, *gm_Flag_SwithcHum=0*. (Fig. 5.4 Pto. 2).

3 y 4 Los puntos de la figura 3 y 4 afectan a la ventilación, y se verán en el apartado correspondiente.

5 Si la humedad desciende del máximo mantener apagado el humificador, *gm_Flag_SwithcHum=0*. (Fig 5.4 Pto. 5).

6 Si la humedad desciende del mínimo programado encender de nuevo el humificador, *gm_Flag_SwithcHum=1*. (Fig 5.4 Pto. 6).

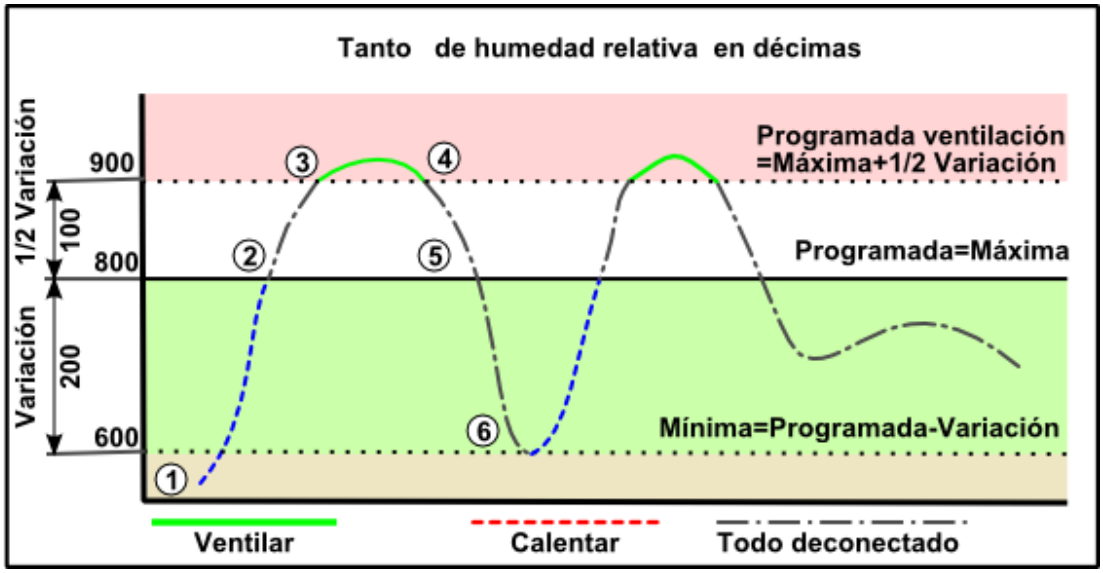


Figura 5.4: Gráfica de lógica de control de humedad

```
void fnc_PRGAUTOHUME(int n)
{
fnc_CalcFlag_SwithcInt(n);
fnc_PRGLimitsTempHum();
if (gm_bActHum >= gm_bActHumPrgMax)
   {
      gm_Flag_SwithcHum = 0;
   }
   else
   {
     if (gm_bActHum <= gm_bActHumPrgMin)
     {
            gm_Flag_SwithcHum = 1;
     }
    }
}
```

fnc_PRGAUTOTEMPHUM(int n): Cálculo del switch de ventilación para el intervalo actual. Tiene por objeto calcular *gm_SwithcTemHum*

que valdrá 1 si se debe encender el relé programado para ventilación, *gm_Flag_SwithcTemHum=0*. (Puntos 4 de las figuras 5.3 y 5.4)

Lógica para encender o apagar el sistema de ventilación

1 Si se sobrepasa la temperatura más la mitad de su variación programada o la humedad máxima más la mitad su variación programada para el intervalo actual, activar ventilación (Renovador de aire), *gm_Flag_SwithcTemHum=1*. (Puntos 3 de las figuras 5.3 y 5.4)

2 Si desciende la temperatura más la mitad de la variación o la humedad máxima más la mitad de la variación programada para el intervalo actual, detener ventilación (Renovador de aire), *gm_Flag_SwithcTemHum=0*. (Puntos 4 de las figuras 5.3 y 5.4)

```
void fnc_PRGAUTOTEMPHUME(int n)
{
  fnc_CalcFlag_SwithcInt(n);
  fnc_PRGLimitsTempHum();
  gm_Flag_SwithcTemHum = 0;
  if (gm_bActTem>(gm_bActTemPrgMax +
  gm_bPrgTemSlotVar*0.5))
    {
       gm_Flag_SwithcTemHum = 1;
    }
      if ( gm_bActHum>(gm_bActHumPrgMax +
      gm_bPrgHumSlotVar*0.5)){
        gm_Flag_SwithcTemHum = 1;}
}
```

TODO 04 Añadir lógica para leer sensores adicionales.

Leer sensor dht22 y almacenar la lectura en variables globales del módulo: *gm_bActHum* (Humedad actual) y *gm_bActTem* (Temperatura actual).

```
//999 Se utiliza como flag de valor anómalo
gm_bActTem = 9999;
gm_bActHum = 9999;
//Leer sensor
gm_bDht22Sts = gm_DHT22.read22(PIN_DHT2);
gm_bActHum = (int)(gm_DHT22.humidity * 10);
gm_bActTem = (int)(gm_DHT22.temperature * 10);
```

TODO 05 Procesar comandos recibidos desde la web del módulo específico.

Cambiar programación de Temperatura.

```
if (command == CMDSETTEM)
{
  //Descomponer los parámetros recibidos como string
```

```
  // URL separados por /
  ptr = strtok( aParameter, " / " );
  // Primera parte CMDSETTEM,
  //no es necesario almacenarla.
  ptr = strtok( NULL, " / " );
  gm_bPrgTemSlotIn = atoi(ptr);
  //Temperatura programada para el intervalo
  ptr = strtok( NULL, " / " );
  gm_bPrgTemSlotOut = atoi(ptr);
  //Temperatura programada fuera del intervalo
  ptr = strtok( NULL, " / " );
  gm_bPrgTemSlotVar = atoi(ptr);
  //Variación permitida hacia abajo.
  g_FlagSave = 1;
  // Indicar que se deben guardar cambios
  // en el fichero de configuración
  // almacenado en microSD
}
```

Cambiar programación de Humedad.

```
if (command == CMDSETHUM)
{
  //Descomponer los parámetros recibidos por el
  //string URL separados por /
  ptr = strtok( aParameter, " / " );
  // Primera parte settime
  ptr = strtok( NULL, " / " );
  gm_bPrgHumSlotIn = atoi(ptr);
  //Humedad programada dentro intervalo
  ptr = strtok( NULL, " / " );
  gm_bPrgHumSlotOut = atoi(ptr);
  //Humedad programada fuera del intervalo
  ptr = strtok( NULL, " / " );
  gm_bPrgHumSlotVar = atoi(ptr);
  //Variación permitida hacia abajo.
  g_FlagSave = 1;
  // Indicar que se deben guardar cambios en el fichero
  // de configuración almacenado en microSD
}
```

TODO 06 Rellenar el buffer específico para el envío a la web, y guardar la configuración.

El buffer específico guarda en un array tanto la temperatura y humedad actual como los valores programados, y se usa posteriormente tanto para guardarlos en el fichero de configuración almacenado en la memoria microSD como para enviarlos a la web y que puedan ser visualizados y cambiados a través de formularios HTML.

```
sprintf (g_BufferMod,
"%04d%04d%04d%04d%04d%04d%04d%04d%01d",
gm_bActTem, gm_bActHum,
gm_bPrgTemSlotIn, gm_bPrgTemSlotOut, gm_bPrgTemSlotVar,
gm_bPrgHumSlotIn , gm_bPrgHumSlotOut,  gm_bPrgHumSlotVar,
gm_bDht22Sts ) ;
```

TODO 07 Rellenar el buffer web completo. No necesita cambios.

TODO 08 Añadir la asignación de valores por defecto a las variables del módulo específico.

Por omisión se obtiene del fichero MyDefines.h los valores por defecto de programación relativos al módulo específico.

```
gm_bPrgTemSlotIn = DEFAULTTEMSLOTIN;
gm_bPrgTemSlotOut = DEFAULTTEMSLOTOUT;
gm_bPrgTemSlotVar = DEFAULTTEMSLOTVAR;
gm_bPrgHumSlotIn = DEFAULTHUMSLOTIN;
gm_bPrgHumSlotOut = DEFAULTHUMSLOTOUT;
gm_bPrgHumSlotVar = DEFAULTHUMSLOTVAR;
```

TODO 09 Obtener los valores de las variables globales del módulo específico a partir del fichero de configuración guardado en la microSD.

Las variables de configuración de temperatura y humedad programadas dentro y fuera del intervalo, así como la variación permitida son leídas desde el fichero almacenado en la microSD con la función *fncReadConfig()* del módulo estándar, aquí solo es necesario descomponerla en sus valores.

```
gm_bPrgTemSlotIn = fncStrToI(68, 72);
gm_bPrgTemSlotOut = fncStrToI(72, 76);
gm_bPrgTemSlotVar = fncStrToI(76, 80);
gm_bPrgHumSlotIn = fncStrToI(80, 84);
gm_bPrgHumSlotOut = fncStrToI(84, 88);
gm_bPrgHumSlotVar = fncStrToI(88, 92);
```

5.4. Modificaciones de software web realizadas para el AR9331 (Linino)

El AR9331 se encarga del interfaz web y de acceder a la tarjeta microSD donde reside el contenido estático de las páginas(HTML,CSS, JavaScript, imágenes...). La parte dinámica(valores de temperatura, humedad, programación, estado de los enchufes...), se obtiene mediante peticiones JavaScript-Ajax al servicio web que se ejecuta en el procesador AR9331-Linux.

En cada página del modulo estándar hay que hacer las siguientes tareas para adaptarlas al modulo específico.

- Agregar en los campos informativos del módulo específico en la página HTML.
- Modificar el Javascript específico de la página para rellenar los campos informativos específicos agregados.
- Agregar al Javascript específico las funciones adicionales para enviar los cambios de configuración a Smartplug01

Nota: En el entorno de desarrollo los ficheros HTML se encuentran en la carpeta: `../smartplug01/web/`, y los Javascript en la carpeta, pero en el kit de Arduino se almacenan en `/mnt/sda1/arduino/www/smartplug01/...`

Los ficheros a modificar y son los siguientes:

Ficheros	Módulo estándar	Módulo específico
index.html y index.js	Muestra la hora, fecha, y programación y estado de los enchufes	Se añade la visualización de la temperatura actual y las nuevas programaciones de los enchufes.
config.html y config.js	Permite cambiar la fecha y hora	Se añade la posibilidad de programar temperatura y humedad para dentro y fuera del intervalo programado en los enchufes.
cfgplugs.html y cfgplugs.js	Permite cambiar la programación de los enchufes	Se añade la posibilidad de de nuevas programaciones en los enchufes, iluminación, calefacción, humificación, ventilación.
guide.html	Breve guía de usuario	Se agrega la explicación de las nuevas opciones.

Tabla 5.2: Lista de ficheros web a modificar

5.4.1. Página web de estado

Los ficheros index.html e index.js se modifican para mostrar la información actual los valores temperatura, humedad y programación de los relés.

5.4.1.1. Página web de estado en HTML (index.html)

En primer lugar programar la página index.html, agregando las etiquetas *span* para recibir desde el AJAX la nueva información:

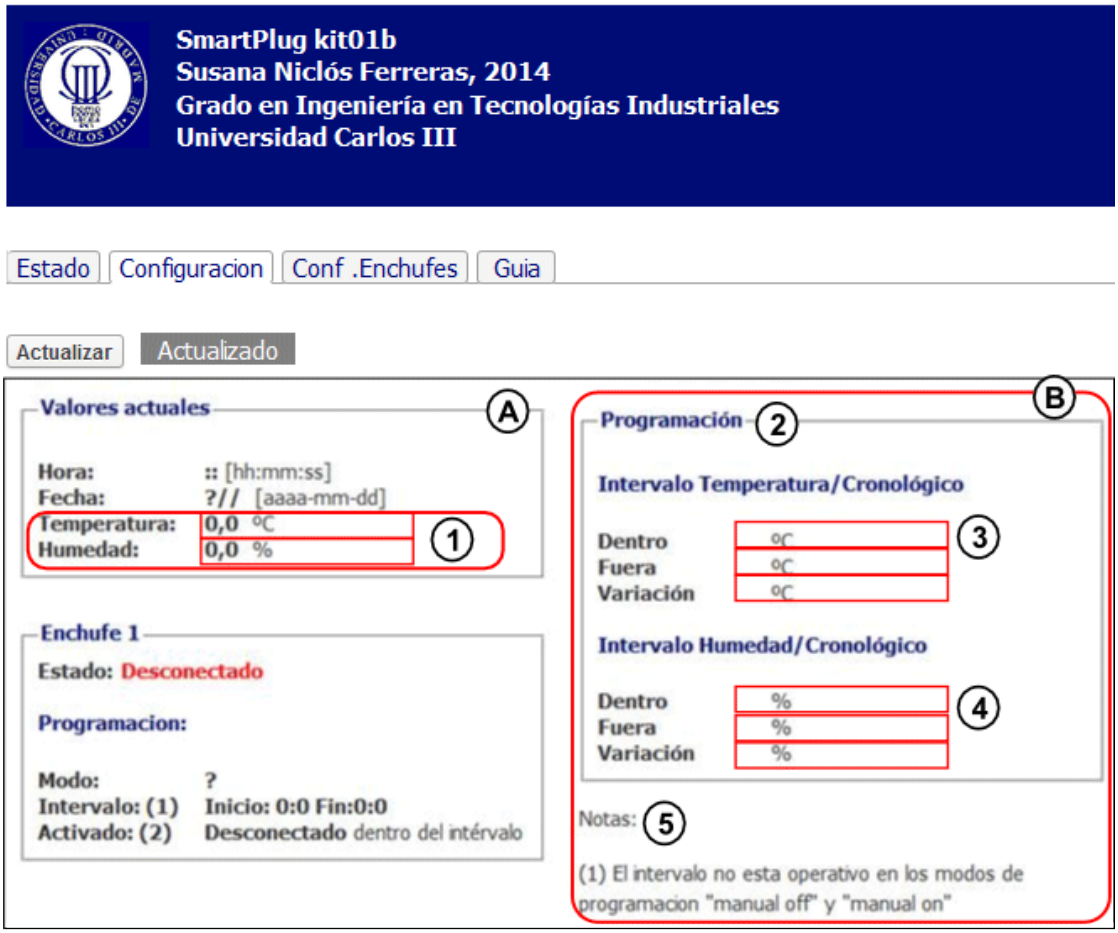


Figura 5.5: Página web de estado

TODO 01 Agregar valores actuales de los sensores añadidos.

En el párrafo "Valores actuales" agregar *span* para mostrar la temperatura y humedad (Fig 5.5. A:1)

Hay que poner especial atención en el nombre *id*, asignado a los *span*, ya que son los que recibirán los valores desde el JavaScript (AJAX), la norma utilizada es que tengan el prefijo "ard" (que indica que se rellenaran de datos recibidos desde el Arduino)

```
<fieldset>
<legend>Valores actuales</legened>
...
<br /><span class="valtitle">Temperatura:</span>
<span id="ardTem" class="valvalue">?</span>  C
<br /><span class="valtitle">Humedad:</span>
<span id="ardHum" class="valvalue">?</span>  %
</fieldset>
```

TODO 02 Agregar programación del módulo específico. Añadir una sección para mostrar los valores programados de temperatura y humedad (B).

Mostrar programación de humedad. (Fig 5.6. B: 2,3,y 4)

```
<fieldset>
<legend>Programación</legened>
<h4><label>Intervalo Temperatura/Cronológico</label></h4>
<br/><span class="valtitle">Dentro</span>
<span id="ardPrgTemIn" class="valvalue ">?</span> C
<br/><span class="valtitle">Fuera</span>
<span id="ardPrgTemOut" class="valvalue ">?</span> C
<br/><span class="valtitle">Variacion</span>
<span id="ardPrgTemVar" class="valvalue ">?</span> C
<h4><label>Intervalo Humedad/Cronológico</label></h4>
<br/><span class="valtitle">Dentro</span>
<span id="ardPrgHumIn" class="valvalue ">?</span> %
<br/><span class="valtitle">Fuera</span>
<span id="ardPrgHumOut" class="valvalue ">?</span> %
<br/><span class="valtitle">Variación</span>
<span id="ardPrgHumVar" class="valvalue ">?</span> %
</fieldset>
```

Incluir nota informativa. (Fig 5.6. B: 5): Detrás del último fieldset opcionalmente se puede agregar una nota informativa, en este caso se ha agregado:

```
<p>Notas:
<p>(1) El intervalo no está operativo en los modos de
 programación "Manual off" y "Manual on"</p>
<p>(2) Esta opción es operativa solo en el caso de
programación horaria, ej: Iluminación"</p>
```

5.4.1.2. Página web de estado en Javascript (index.js)

TODO 01 Leer los campos específicos, descomponiendo el buffer.

```
var vActTemp = miArray.substring(60,64);
var vActHum = miArray.substring(64,68);
var vPrgTemSlotIn = miArray.substring(68,72);
var vPrgTemSlotOut = miArray.substring(72,76);
var vPrgTemVariation = miArray.substring(76,80);
var vPrgHumSlotIn = miArray.substring(80,84);
var vPrgHumSlotOut = miArray.substring(84,88);
var vPrgHumVariation = miArray.substring(88,92);
```

TODO 02 Mostrar campos del módulo específico (humedad y temperatura).

```
var     iActTemFul=1*vActTemp;
var IiActTemEnt=Math.floor((1*iActTemFul)/10);
var iActTemDec=iActTemFul % 10  ;
$('#ardTem').text(''+iActTemEnt+','+iActTemDec);
var     iActHumFul=1*vActHum;
var iActHumEnt=Math.floor((1*iActHumFul)/10);
var iActHumDec=iActHumFul % 10  ;
$('#ardHum').text(''+iActHumEnt+','+iActHumDec);
```

TODO 03 Agregar mensajes de relé en función del modo de programación del módulo específico.

Los modos de programación se reciben del Arduino con un identificador numérico, y se muestran en la página como texto, para ello se utiliza esta matriz.

```
var aMsg=[
"0 Manual OFF",
"1 Manual ON",
"2 Automático. Intervalo horario \(Slot\)".
"3 Automático. En función de temperatura
 e intervalo horario. (Calefactor)".
"4 Automático. En función de humedad
 e intervalo horario. (Humificador)",
"5 Automático. En función de  humedad y temperatura
 e intervalo. (Ventilación)"]
```

5.4.2. Página web de configuración de temperatura, humedad y reloj

Config.html y config.js son los ficheros que se modifican para permitir además de cambiar la fecha y hora del reloj, y la programación en el intervalo, poder cambiar la variación de la temperatura y de la humedad.

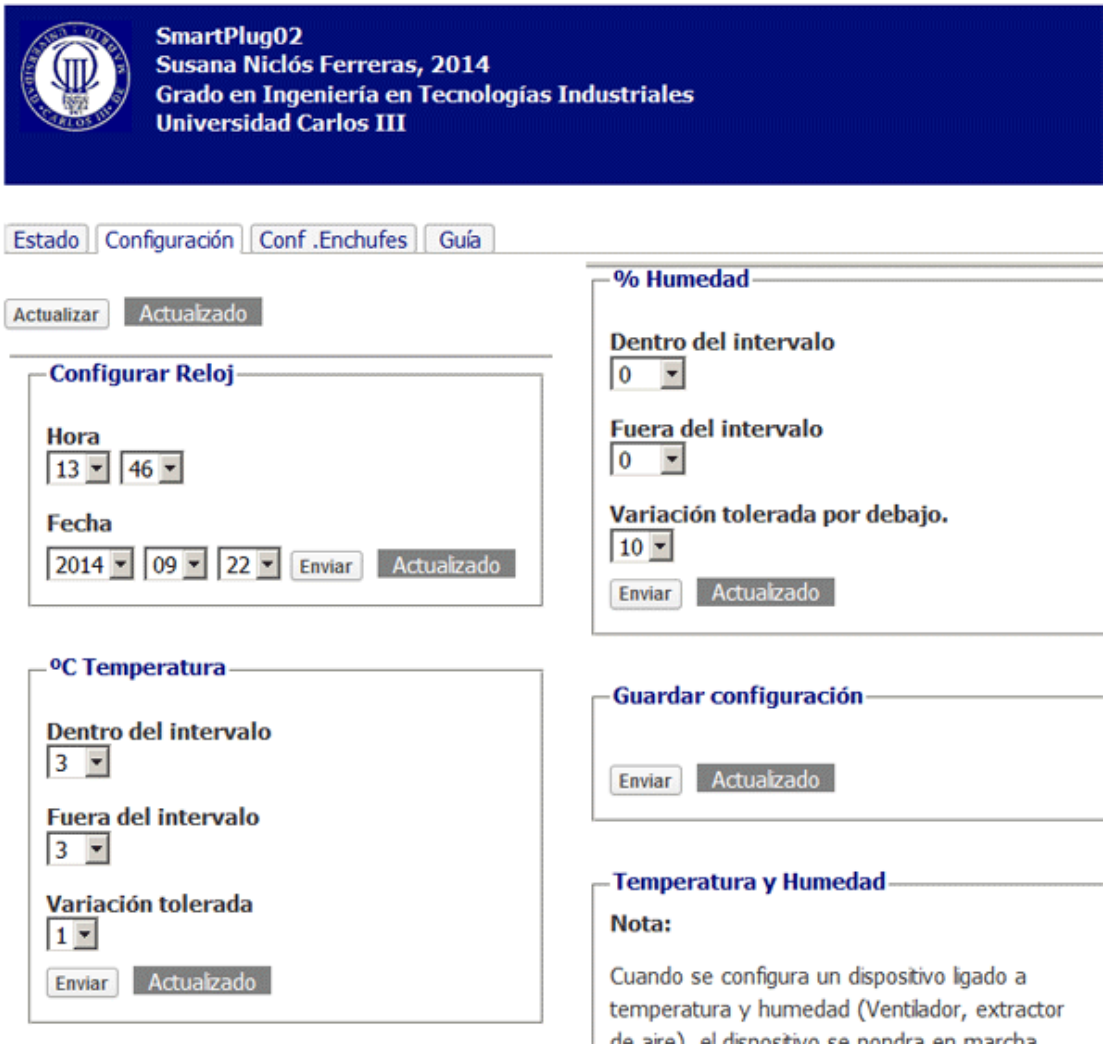


Figura 5.6: Configuración de temperatura, humedad y reloj

5.4.2.1. Página web de configuración en HTML (config.html)

En primer lugar programar la página config.html, agregando las etiquetas *span*, los campos de entrada de datos y botones de envío de comandos:

Campos y botones para enviar la configuración de temperatura:

- **PrgTemOut:** Temperatura dentro del intervalo del relé.
- **PrgTemInt:** Temperatura fuera del intervalo del relé.
- **PrgTemVar:** Variación de temperatura permitida hacia abajo.
- **Botón enviar:** Llamar al función *fncSetTem()* Javascript, que mediante AJAX envía la configuración.

```
<fieldset>
<legend>C Temperatura </legend>
<label for="prgTemIn"  class="valtitle3" >
Dentro del intervalo</label>
<select id="prgTemIn" name="prgTemIn">
   <option value="03">3</option>
   ...
   <option value="32">32</option>
</select>
<br/><label for="prgTemOut"  class="valtitle3" >
Fuera del intervalo</label>
<select id="prgTemOut" name="prgTemOut">
   <option value="03">3</option>
   ...
   <option value="32">32</option>
</select>
<br/><label for="prgTemVar"  class="valtitle3" >
Variación tolerada</label>
<select id="prgTemVar" name="prgTemVar">
<option value="01">1</option>
   ...
   <option value="04">4</option>
</select>
<br /><input type="button" class="btnmini"
value="Enviar" onclick="fncSetTem();"/>
<span id="ardMsgTem">Por favor espere</span>
</fieldset>
```

Campos y botones para enviar la configuración de humedad

- **PrgHumOut:** Humedad dentro del intervalo del relé.
- **PrgHumInt:** Humedad fuera del intervalo del relé.
- **PrgHumVar:** Variación de humedad permitida hacia abajo.
- **Botón enviar:** Llamar a la función fncSetHum(), Javascript que mediante AJAX envía la configuración.

```
<fieldset>
<legend>Humedad</legend>
<label for="prgHumIn"  class="valtitle3" >
Dentro del intervalo</label>
<select id="prgHumIn" name="prgHumIn">
   <option value="000">0</option>
   <option value="020">20</option>
   ...
   <option value="100">100</option>
</select>
```

```
<br/><label for="prgHumOut"  class="valtitle3" >
Fuera del intervalo</label>
<select id="prgHumOut" name="prgHumOut">
   <option value="000">0</option>
   <option value="010">10</option>
   ...
   <option value="100">100</option>
</select>
<br/><label for="prgHumOutVar"  class="valtitle3">
Variacion tolerada por debajo.</label>
<select id="prgHumVar" name="prgHumOutVar">
   <option value="010">10</option>
   ...
  <option value="030">30</option>
</select>
<br /><input type="button" class="btnmini"
value="Enviar" onclick="fncSetHum();"/>
<span id="ardMsgHum" >Por favor espere</span>
</fieldset>
```

5.4.2.2. Página web de configuración en Javascript (config.js)

TODO 01 Leer variables específicas.

```
var vActTemp = miArray.substring(60,64);
var vActHum = miArray.substring(64,68);
var vPrgTemSlotIn = miArray.substring(68,72);
var vPrgTemSlotOut = miArray.substring(72,76);
var vPrgTemVariation = miArray.substring(76,80);
var vPrgHumSlotIn = miArray.substring(80,84);
var vPrgHumSlotOut = miArray.substring(84,88);
var vPrgHumVariation = miArray.substring(88,92);
```

TODO 02 Mostrar los valores en pantalla.

```
document.getElementById("prgTemIn").value=vPrgTemSlotIn;
document.getElementById("prgTemOut").value=vPrgTemSlotOut;
document.getElementById("prgTemVar").value=vPrgTemVariation;
document.getElementById("prgHumIn").value=vPrgHumSlotIn;
document.getElementById("prgHumOut").value=vPrgHumSlotOut;
document.getElementById("prgHumVar").value=vPrgHumVariation;
```

TODO 03: Funciones adicionales encargadas de enviar la configuración de temperatura y humedad al servidor.

Función para cambiar configuración la temperatura:

```
function fncSetTem()
```

```
{
//#define CMDSETTEM "3"
        $('#ardMsgTem').text('Cambiando temperatura');
        var IdURL='/arduino/3/'+prgTemIn.value+'/'+
        prgTemOut.value+'/'+prgTemVar.value;
        $.get(IdURL);
        setTimeout(fncGetRefresh, 3000)
}
```

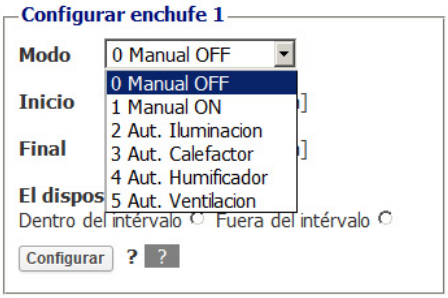
Función para cambiar la configuración de la humedad:

```
// Configurar humedad para el intervalo y fuera de él
function fncSetHum()
{
//#define CMDSETHUM "4"
  $('#ardMsgHum').text('Cambiando humedad');
  varIdURL='/arduino/4/'+prgHumIn.value+'/'+
  prgHumOut.value+'/'+prgHumVar.value;
  alert(IdURL);
  $.get(IdURL);
  setTimeout(fncGetRefresh, 3000)
}
```

5.4.3. Página web de configuración de la programación de enchufes

Cfgplugs.html y cfgplugs.js son los ficheros a modificar.

Solo es necesario agregar a los campos select los nuevos tipos de programación: Calefactor, humificador, ventilador y reutilizar el modo 2 para iluminación.




```
<select id="prgModeRL1">
   <option value="0">0 Manual OFF</option>
   <option value="1">1 Manual ON</option>
   <option value="2">2 Aut. Iluminacion </option>
   <option value="3">3 Aut. Calefactor</option>
   <option value="4">4 Aut. Humificador</option>
   <option value="5">5 Aut. Ventilacion</option>
</select>
```

Figura 5.7: Modos de programación de los enchufes

5.4.3.1. Página de configuración de enchufes en HTML (cfgplugs.html)

TODO 01 Agregar modos de programación para el relé 1, localizar el campo select titulado *prgModeRL0* y agregar los cambios de modo de programación.

```
<legend>Configurar enchufe 2</legend>
<span class="valtitle2">Modo</span>
<select id="prgModeRL1">
   <option value="0">0 Manual OFF</option>
   <option value="1">1 Manual ON</option>
   <option value="2">2 Aut. Iluminación </option>
   <option value="3">3 Aut. Calefactor</option>
   <option value="4">4 Aut. Humificador</option>
   <option value="5">5 Aut. Ventilación</option>
</select>
```

TODO 02, 03 y 0 Siguientes relés. Repetir el proceso anterior respecto de los campos select titulados *prgModeRL1*, *prgModeRL2* y *prgModeRL3*.

5.4.3.2. Página configuración de enchufes en Javascript (cfgplugs.js)

No se necesitan realizar cambios.

5.4.4. Página web guía de usuario

Esta página tiene por objeto enseñar al usuario el manejo de la interfaz web. El programador de un nuevo módulo adaptado, debe modificar el contenido para adaptarlo a sus necesidades. Esta página no necesita una configuración Javascript especial.

A pie de página contiene enlaces para descargar la memoria y los códigos fuente del proyecto.

Figura 5.8: Guía de usuario

5.4.4.1. Página web guía de usuario en HTML (guide.html)

TODO 01 Modificar el contenido. Dentro de la etiqueta, cambiar el contenido según las necesidades. Para mantener un estilo coherente se indica como hacer el índice y sus secciones.

```
<h4>Índice</h4>
<ul>
<li><a href="sec01">Página de información Index.html</a></li>
<li><a href="sec01">Acceder a la configuración de red wifi
</a></li>
</ul>
<h2><a name="sec01">Página de información Index.html
</a></h2>
...
<h2><a name="sec01">Acceder a la configuración de red wifi
</a></h2>
...
```

Capítulo 6

Guía para crear nuevas aplicaciones 2: módulo de sistemas de alarma, SmartPlug02

El objetivo del módulo de sistemas de alarma es crear, partiendo del estándar un sistema de seguridad pasiva, con la capacidad de detectar fugas de gas e intrusos, y la capacidad de avisar acústicamente, visualmente y por email en caso de detectarlos. Además se quiere que sirva de guía para mostrar como integrar el uso de Python y el envío correo electrónico sobre el módulo estándar.

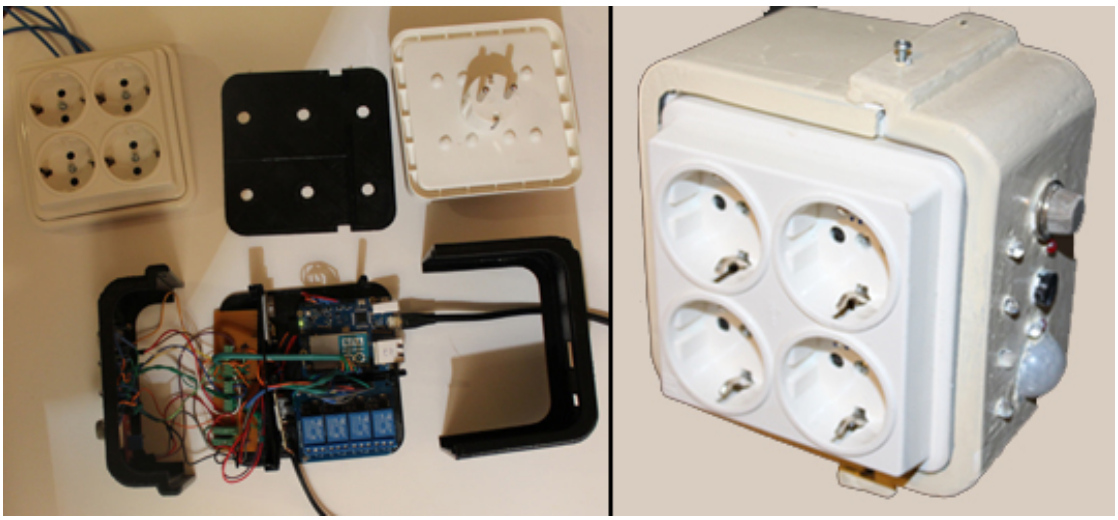


Figura 6.1: SmartPlug02 piezas y montado

Funcionalidades añadidas o reutilizadas.		
Funcionalidad	Módulo estándar	Módulo Alarmas
Servidor Web	✓	✓
Configuración programación mediante Web	✓	✓(Ampliada)
Acceso Wifi Internet	✓	✓
4 Relés que controla 4 enchufes	✓	✓
Modo programación Manual off	✓	✓
Modo programación Manual on	✓	✓
Modo programación por intervalos de tiempo		✓
Modo programación por alarma de presencia		✓(Alarma PIR, activar enchufe, enviar email, emitir zumbido.)
Modo programación de alarma de gas		✓(Alarma gas, activar enchufe, enviar email, sonar zumbador.)
Servicio envío de correos		✓(Formulario HTML para introducir la configuración de envío de email de alerta. Script en Python para el envío de alertas.)

Tabla 6.1: Comparativa del módulo estándar y de alarmas

6.1. Hardware añadido y conexiones

Se ha añadido tan solo un sensor de gas y un PIR.

6.1.1. Sensor de movimiento PIR

El PIR permite detectar el movimiento en un rango de 7 metros (120º) [22]. Consta de un sensor piroeléctrico capaz de detectar niveles de radiación infrarroja conectado de tal forma que solo detecta las variaciones, ya que solo nos interesan los cambios. Además el PIR cuenta con el chip BISS0001 que procesa la señal analógica y la convierte en digital.

Este módulo puede ser calibrado para ajustar el tiempo que tarda en dispararse la señal y la sensibilidad, es barato, de poco consumo y fácil de usar.

Para aumentar el angulo de detección el PIR cuenta con una lente de Fresnel que condensa la luz aumentando el rango de detección del sensor infrarrojo a 120º.

6.1.2. Sensor de gas MQ5

El MQ5 es capaz de detectar concentraciones de gas entre 200-10000ppm, tiene una alta sensibilidad ante GLP y gas natural, y baja ante humo [21].

Este sensor puede ser calibrado para distintos gases y concentraciones variando el valor de la resistencia. Normalmente se utiliza una resistencia de entorno a 20k, aunque según varíen las condiciones de temperatura y humedad este valor se puede ver influenciado.

Nota: Es importante resaltar que este sensor necesita un periodo de calentamiento de 20 segundos para proporcionar lecturas fiables. También se ha detectado que se dispara ante la presencia de algunos aerosoles y limpia cristales (Alcoholes).

6.1.3. Avisos visuales y acústicos.

1. **Led verde:** Ligado al PIR, se ilumina cuando hay movimiento.
2. **Led rojo:** Ligado al PIR, se ilumina cuando se ha detectado movimiento, y la alarma esta conectada. Se apaga manualmente.
3. **Led rojo:** Ligado al MQ5. Se ilumina cuando hay presencia de gas.
4. **Zumbador:** Se reutiliza el del módulo estándar y se usa para emitir una señal intermitente cuando se dispara la alarma de gas o la de presencia.

6.1.4. Conexiones

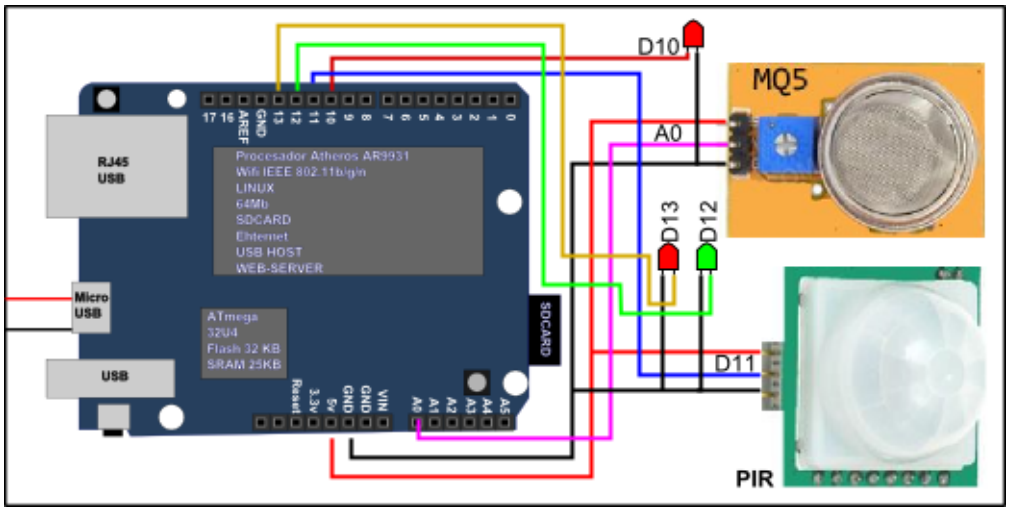


Figura 6.2: Esquema de conexiones del Smartplug02

Dispositivo	PIN	Función
PIR	D11	Pin de datos
LED verde	D12	Led movimiento
LED rojo	D13	Alarma movimiento
MQ5	A0	Sensor de gas, pin de datos
Led rojo	D10	Alarma, gas detectado

Tabla 6.2: Conexiones SmartPlug02.

6.2. Software, creación del nuevo proyecto

Como en el capítulo anterior son necesarias modificaciones de software tanto en la parte realizada en C++ destinada al ATmega32U4 como la parte desarrollada para la web realizada en HTML y Javascript que utiliza el procesador AR9331(Linino-Linux), en la que además se han desarrollado dos Script en Python.

6.2.1. Creación de la base para el nuevo proyecto. SmartPlug02

Al igual que se realizó en el capitulo anterior, se parte del módulo estándar denominado Smartplug00, y sobre él se desarrolla el nuevo proyecto que titulamos SmartPlug02.

1. Crear una carpeta con el nombre Smartplug02, que contendrá el nuevo proyecto.
2. Copiar el contendido de la carpeta Smartplug00 en la carpeta Smartplug02
3. Situarse en la carpeta Smartplug02 y redenominar el archivo smartplug**00**.ino a smartplug**02**.ino

TODO: Como se ha explicado en el módulo anterior, el código original contiene líneas cuyo comentario comienza por TODO, para indicar los lugares donde se deben realizar los cambios.

6.3. Modificaciones de software realizadas para el ATmega32U4

No son necesarias librerías adicionales, solo hay que modificar los distintos ficheros del programa añadiendo tras el TODO el código pertinente.

6.3.1. Proyecto Smartplug02 modificación fichero MyDefines.H

En este fichero agregar las constantes tipo defines usadas para declarar pines, modos de programación y comandos web, etc.

TODO 01 Cambiar el nombre del ficheros de configuración.

```
#define CONFIGFILENAME "/mnt/sd/smartplug02.cfg"
```

TODO 02 Agregar a partir de aquí las constantes para pines específicos

```
// Pin de datos de gas
#define PIN_GASSNS 0
// Pin led rojo,  indica presencia de gas
#define PIN_GASLED 10
// Pin de datos del PIR
#define PIN_PIRSNS 11
// Pin led verde, indica presencia detectada por el PIR
#define PIN_PIRLEDMOV 12
// Pin led rojo, indica alarma sonando
#define PIN_PIRLEDALA 13
```

TODO 03 Agregar a partir de aquí las constantes que se usan para modos de programación de los relés

```
#define PRGAUTOGAS  3
#define PRGAUTOPIR  4
```

TODO 04 Agregar a partir de aquí valores por defecto

```
// Por defecto alarma desconectada
#define PRGDEFAULTPIRALARMODE=0
```

TODO 05 Agregar a partir de aquí los identificadores de comandos web.

```
// Desactivar alarma de presencia (Sonido)
#define CMDSETPIRALARMOFF "3"
// Activar alarma de presencia (Sonido)
#define CMDSETPIRALARMON  "4"
// Configurar dirección de correo para envío de alarmas.
#define CMDSETMAIL "5"
```

TODO 06 Agregar a partir de aquí otros valores.

```
// Espera para que el sensor se calibre y sincronizar arranque linux
#define CALIBRATIONGASMILLIS 60000
// Nivel por debajo del cual salta la alarma del gas
#define GASLEVEL 150
// Frecuencia de la alarma
#define TONEFREC 880
// Duración de cada pulso de la alarma
#define TONEMILLIS 880
```

6.3.2. Proyecto Smartplug02 modificaciones fichero smartplug02.ino.

TODO 01 Includes. No son necesarios modificaciones.

TODO 02 A Agregar nuevas variables globales. Se aconseja utilizar el prefijo. **gm**_ (global module) para evitar posibles redefiniciones de variable y facilitar la lectura del código.

```
byte    gm_bGasAlarmBeeping = 0;
// 0= Alarma desactivada, 1= Alarma activa.
byte    gm_FlagPirAlarmFiringMode = 0;
// Solo se activa si esta en modo alarma activa y se
// ha disparado la alarma
byte    gm_FlagPirAlarmBeeping = 0  ;
// 0= Alarma en disparada, 1= Alarma no disparada
byte    gm_FlagPirAlarmFiringlarmBeeping = 0;
```

TODO 02 B En la función *Setup()*del módulo específico antes de inicializar el *Bridge* añadir la declaración de puertos.

```
pinMode(PIN_PIRSNS, INPUT);
pinMode(PIN_PIRLEDMOV, OUTPUT);
pinMode(PIN_PIRLEDALA, OUTPUT);
// Por omisión no hay movimento
digitalWrite(PIN_PIRLEDMOV, LOW);
pinMode(PIN_GASSNS, INPUT);
pinMode(PIN_GASLED, OUTPUT);
 // Por omisión la alarma no suena
digitalWrite(PIN_GASLED, LOW);
\item[TODO 02 B FIN]
```

TODO 02 C En la función *Setup()* después de iniciado el *Bridge* agregar equipo adicional.

```
Bridge.put("emp", "0"); // No enviar mail PIR
Bridge.put("emg", "0"); // No enviar mail gas
```

TODO 03 Agregar al switch que controla los modos de programación estándar las funcionalidades adicionales. Al leer los sensores se obtienen los valores de las variables *gm_bPirAlarmBeeping* y *gm_bGasAlarmBeeping*, que toman el valor 1 si la alarma esta sonando.

```
case  PRGAUTOPIR:
g_bOnOff=gm_bPirAlarmBeeping;
case  PRGAUTOGAS:
g_bOnOff=gm_bGasAlarmBeeping;
```

TODO 04 Procesar comandos recibidos desde la web del módulo específico.

```
//Activar y desactivar alarma de movimiento
if (command == CMDSETPIRALARMOFF)
{
// Flag para evitar el envío de email
Bridge.put("emp", "0");
gm_bPirAlarmBeeping = 0;
gm_bPirAlarmFiringMode = 0;// Desconectada
client.println( 3);
fncConfigSave();
return;
}
if (command == CMDSETPIRALARMON)
{
gm_bPirAlarmFiringMode = 1; // Conectada
client.println( 2);
fncConfigSave();
return;
}
```

TODO 05 Añadir la lógica para leer sensores adicionales. Leer el sensor de alarma y el gas, cambiar variables globales específicas, y disparar alarmas.

```
noTone(PIN_BUZZER);// Silenciar zumbador.
// ========================================
// Sección PIR Sensor de movimiento
digitalWrite(PIN_PIRLEDMOV, digitalRead(PIN_PIRSNS));
if (digitalRead(PIN_PIRSNS) == HIGH &&
gm_bPirAlarmFiringMode == 1){gm_bPirAlarmBeeping = 1;}
// Disparar solo si está conectada.
// Esto es así porque la alarma puede haberse disparado
// en un ciclo previo del programa, y debe seguir
// sonando hasta que se desconecte por web.
if (gm\_bPirAlarmBeeping == 1)
{
   digitalWrite(PIN_PIRLEDMOV , HIGH);
   // Ordenar enviar un email al proceso sendmail.py
   Bridge.put("emp", "1");
```

```
    // Sonar un beep de alerta
    tone(PIN_BUZZER, TONEFREC, TONEMILLIS);
  }
  else
  {
    // Apagar led movimiento
    digitalWrite(PIN_PIRLEDALA, LOW);
  }
  // ========================================
  // Sección sensor de gas
  if  (analogRead (PIN_GASSNS) > GASLEVEL)
    {
      // Apagar alarma gas
      digitalWrite(PIN_GASLED, LOW);
      gm_bGasAlarmFiring = 0;
      Bridge.put("emg", "0");
    }
    else
    {
      // Encender alarma gas
      digitalWrite(PIN_GASLED, HIGH);
      // Ordenar enviar email al proceso sendmail.py
      Bridge.put("emg", "1");
      gm_bGasAlarmFiring = 1;
      // Sonar un beep de alerta
      tone(PIN_BUZZER, TONEFREC , TONEMILLIS);
    }
```

TODO 06 Rellenar el buffer específico para el envío a la web, y guardar la configuración.

El buffer específico contiene en un array las variables que representan: si la alarma PIR esta activa, si se detecta movimiento, si esta sonando, y si la alarma de gas esta sonando. Este buffer se usa para guardar el fichero de configuración situado en la microSD, y para el envío de datos a la web.

```
  sprintf (g_BufferMod, "%01d%01d%01d%01d"
  gm_bPirAlarmFiringMode,digitalRead(PIN_PIRSNS),
  gm_bPirAlarmBeeping, digitalRead(PIN_GASLED));
```

TODO 07 Rellenar el buffer web completo. No necesita cambios.

TODO 08 Añadir los valores por defecto del módulo especifico.

Por omisión estos datos se obtienen del fichero MyDefines.h, pero hay que asignarlos a las variables globales.

```
gm_bPirAlarmFiringMode=PRGDEFAULTPIRALARMODE;
```

TODO 09 Obtener los valores de las variables globales del módulo específico a partir del fichero de configuración guardado en la microSD.

Permite que después de reiniciar el equipo la alarma vuelva al estado en que se configuró previamente.

```
gm_bPirAlarmFiringMode = fncStrToI(60, 60);
```

6.4. Modificaciones de software web realizadas para el AR9331 (Linux).

El software de la web es casi igual que en el ejemplo anterior, a excepción de que en este caso el servidor linux también se ha configurado para interpretar peticiones URL que llaman a un Script de Python. Con ellas se envían emails y se guardan los datos de configuración de la cuenta de correo.

El Script de Python encargado del envío de emails es el proceso sendmail.py, este por defecto utiliza cuentas de gmail y envía los siguientes correos alerta:

- **Email disparo de alarma de presencia:**

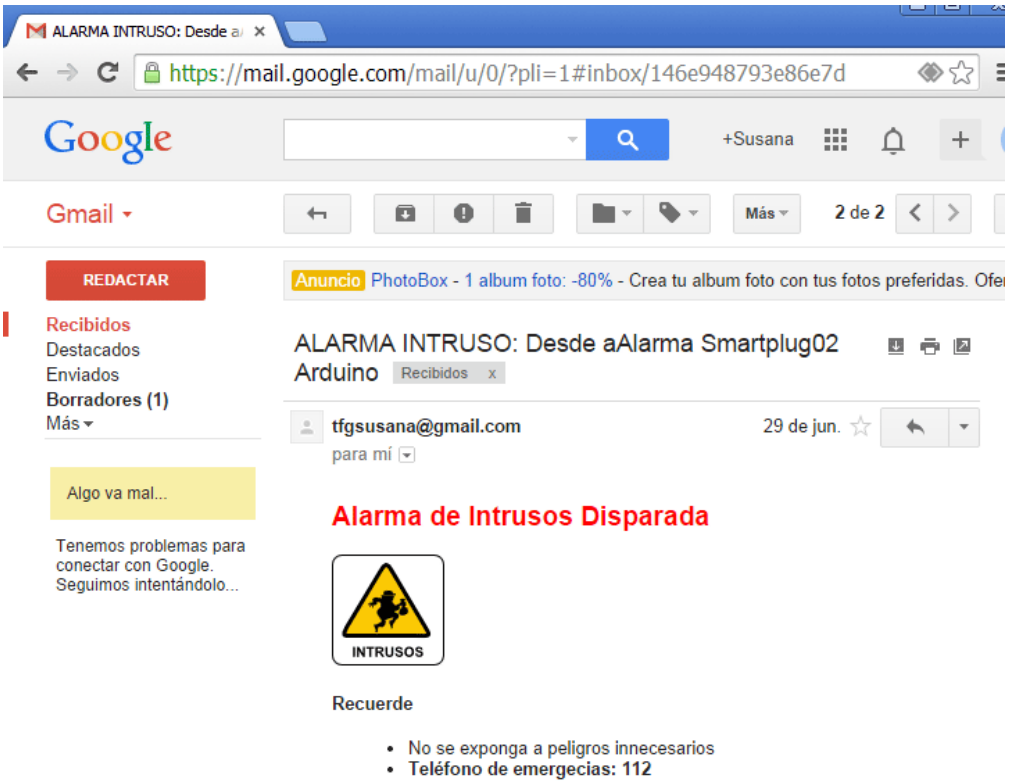


Figura 6.3: Email de alerta por disparo de alarma de presencia

- **Email disparo de alarma de gas:**

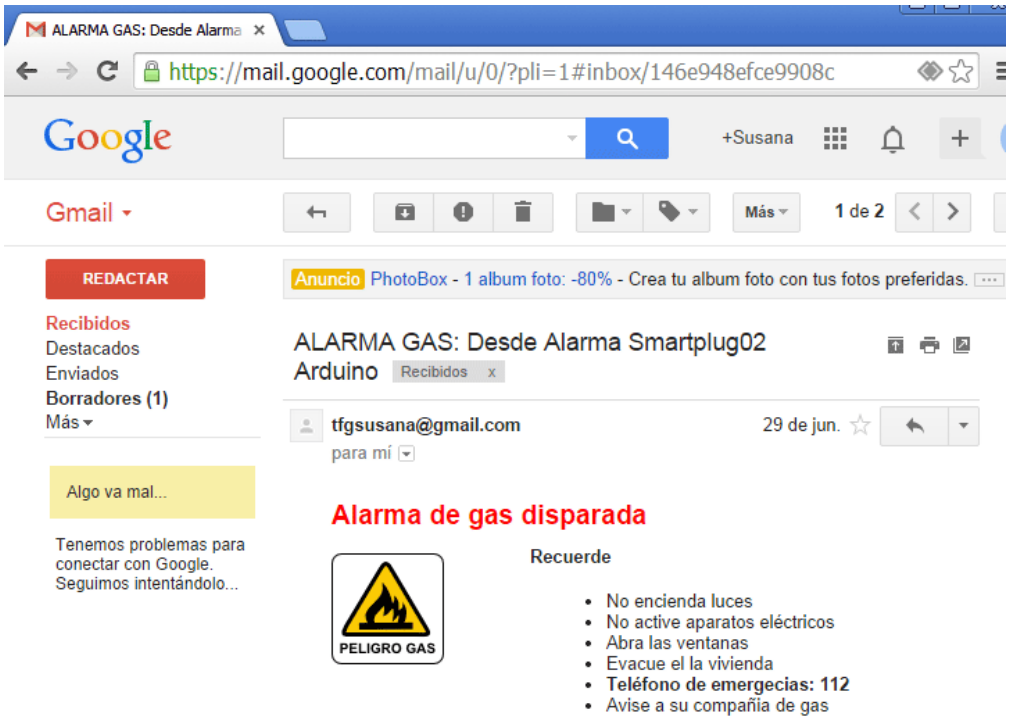


Figura 6.4: Email de alerta por disparo de alarma de gas

6.4.1. Procesos y ficheros involucrados en el envío de emails de alerta.

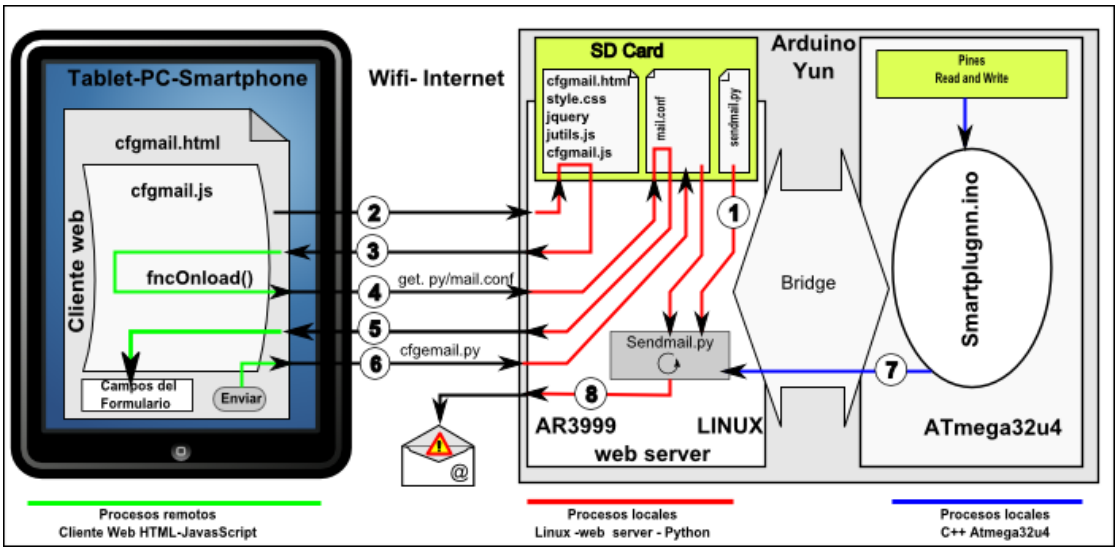


Figura 6.5: Procesos asociados al envío de emails

6.4.1.1. Circulación de la información relativa al envío de email

A continuación se explica como se produce el intercambio de datos entre los procesos relacionados con el envío de emails, según el ejemplo representado en la figura 6.5.

1 Cargar sendmail.py: Es el proceso encargado de enviar emails. Al iniciarse el Arduino, el proceso arranca quedando residente en la memoria.

El proceso lee cíclicamente mediante el Bridge el valor de las variables *emg* (email gas) y *emp* (email pir) y en el caso de que su valor sea 1, envía el email correspondiente.

Nota: El valor de las variables *emp* y *emg* lo modifica el proceso C++ que se ejecuta el el procesador ATmega32U4, y que es el único que tiene acceso a los sensores.

2 Pedir la página web de configuración de correo: Cuando el usuario quiere configurar la cuenta de correo solicita la página `http://../smartplug02/cfgmail.html`.

3 Recibir la página: En su dispositivo se recibe la página estática y los ficheros complementarios: JavaScript, CSS...

4 Pedir los datos de última configuración de correo: Una vez recibida toda la página estática y ficheros complementarios se lanza el proceso *fncOnLoad()* que solicita mediante ajax los datos de la última configuración para poder rellenar la información.

5 Recibir datos y rellenar el formulario: Cuando se ha recibido el fichero mail.conf, el JavaScript rellena los campos usuario, contraseña... sobre la página estática, permitiendo su posterior modificación.

6 Enviar datos de configuración: Si se desea modificar los datos de configuración, el usuario rellena la nueva información sobre la página y se hace click en el botón enviar. Así los datos se envían mediante una petición AJAX, que a diferencia del caso general solicita ser atendida por un script en Python. Este se encarga de recibir los datos modificados y grabarlos en la microSD, sin necesidad de que intervenga el proceso C++.

7 Ordenar envío de un email de alerta: Las ordenes de mandar emails provienen del procesador ATmega32U4. En este microprocesador se ejecuta el programa en C++ encargado de leer los sensores y disparar las alarmas. Si la alarma gas es disparada, utilizando el *Bridge* el ATmega32U4 pone a 1 el valor de la variable *emg*, y si la alarma de presencia se dispara, *emp* toma el valor 1.

8 Enviar correo: Si las variables *emg* o *emp* valen uno, el proceso sendmail.py lee la última configuración de correo, se conecta al servidor, y envía por email el mensaje de alerta correspondiente.

6.4.1.2. Proceso sendmail.py

Este Script Python se carga una vez al iniciar el Smartplug, y permanece activo todo el tiempo. En su bucle principal lee periódicamente por medio del *Bridge* los valores *emg* (Flag para email gas) y *emp* (Flag para email pir), en el caso de que su valor sea 1, se dispara el proceso de envío de email.

El diagrama de flujo de la función es el siguiente:

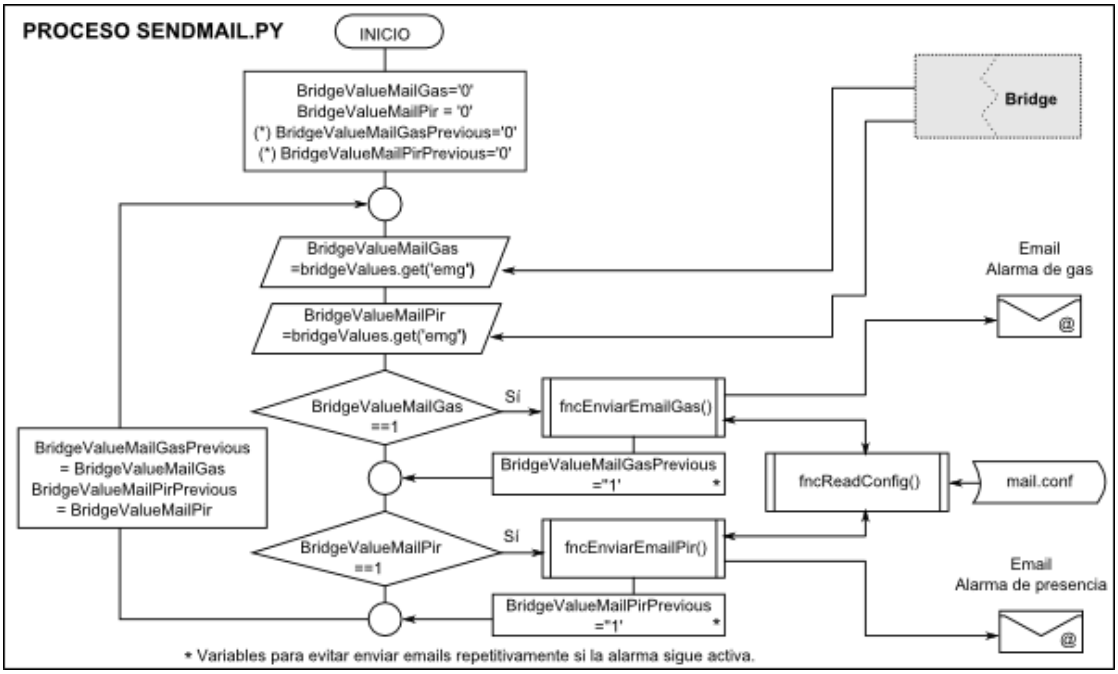


Figura 6.6: Diagrama de flujo función sendmail.py

Nota: A la estructura de carpetas del entorno de desarrollo web se le ha agregado una que contiene los ficheros escritos en Python. Así las capetas quedan de la siguiente forma:

1. **Páginas web:** ...<smartplug02 >\www \
2. **JavaScripts:** ...<smartplug02 >\www \js
3. **Hojas de estilo:** ...<smartplug02 >\www \css
4. **Pyton:** ...<smartplug02 >\www \py

6.4.2. Resumen de modificaciones en la web

Las modificaciones de la web han de seguir la mismas pautas que en ejemplo anterior: Agregar etiquetas *span* a ficheros HTML para contener los campos informativos nuevos y modificar Javascript para manejarlos.

Los ficheros a modificar o a agregar son los siguientes:

Ficheros	Módulo estándar	Módulo específico.
index.html y index.js	Muestra la hora, fecha, y programación y estado de los enchufes.	Se añade la visualización de la temperatura actual y las nuevas programaciones de los enchufes.
config.html y config.js	Permite cambiar la fecha y hora.	Se añade la posibilidad de programar temperatura y humedad para dentro y fuera del intervalo programado en los enchufes.
cfgplugs.html y cfgplugs.js	Permite cambiar la programación de los enchufes.	Se añade la posibilidad de de nuevas programaciones en los enchufes, ilumina-ción, calefacción, humifica-ción, ventilación.
guide.html	Breve guía de usuario.	Se agrega la explicación de las nuevas opciones.
web Menu		En todas las páginas se agrega la opción configurar correo.
cfgemail.html		Configurar datos para el envío de alarmas por email.
cfgemail.py		Script Python para guardar la configuración de email en la microSD.
sendmail.py		Script Python para enviar email.

Tabla 6.3: Lista de ficheros web a modificar

6.4.3. Configurar menú web superior

En todas las páginas HTML hay que modificar el menú de pestañas para que aparezca la opción configurar email

```
<ul class="tabs">
<li  class="active"><a href="index.html"
title="Estado de alarma y enchufes" >Estado</a></li>
<li><a href="config.html"
title  ="Configurar Alarma y Reloj">Configuración</a></li>
<li><a href="cfgplugs.html"
title="Configurar enchufes">Conf. Enchufes</a></li>
<li><a href="cfgemail.html"
title="Configurar email">Conf. Email</a></li>
<li><a href="guide.html">Guía</a></li></ul>
```

6.4.4. Página web de estado.

Index.html y index.js son los ficheros que se han modificado para mostrar la información actual de la lectura de los sensores de gas, movimiento, reloj, estado de los relés y programación.

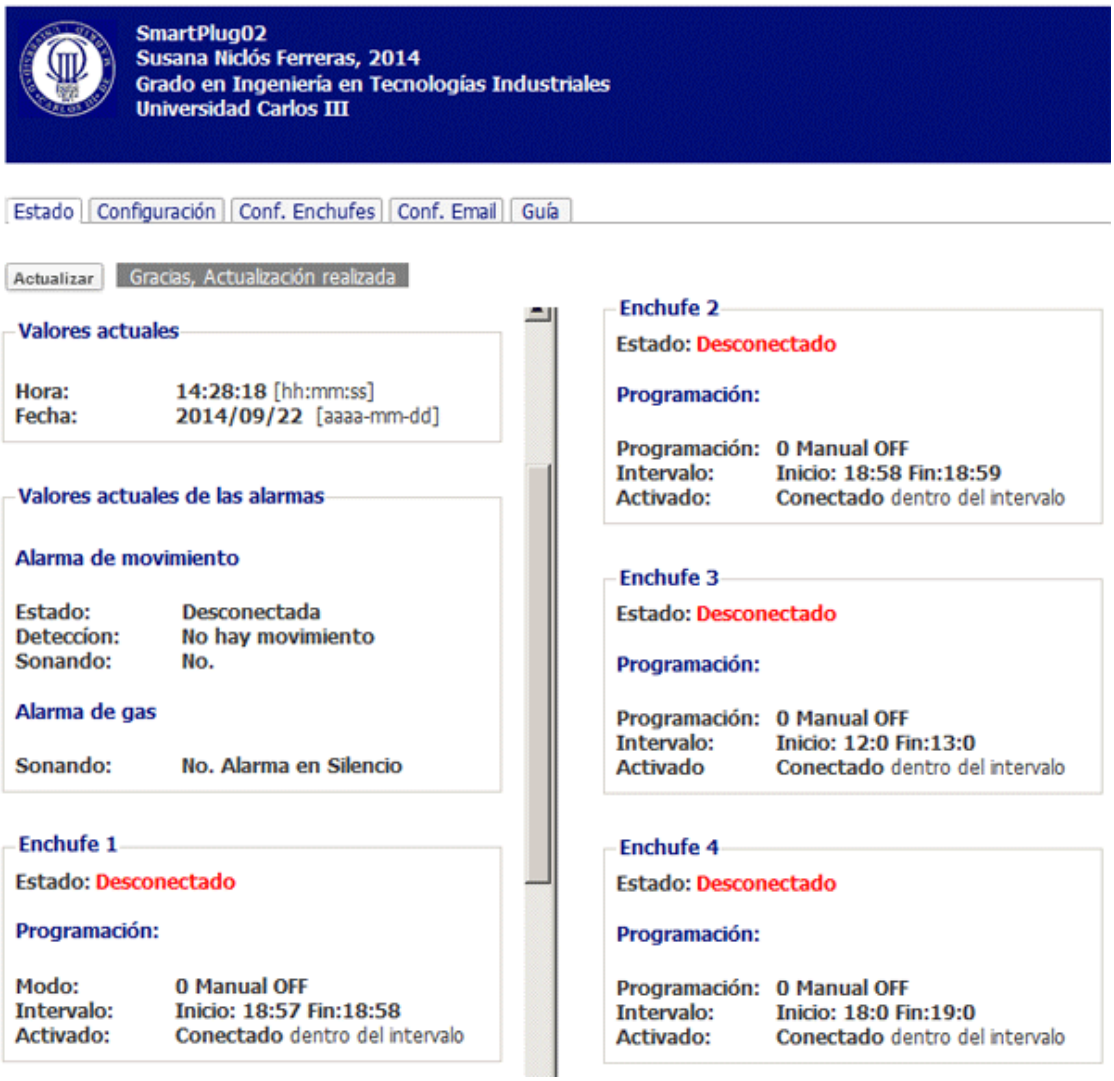


Figura 6.7: Campos adicionales del formulario index.html.

6.4.4.1. Página web de estado en HTML (index.html)

En primer lugar programar la página index.html, agregando las etiquetas *span* para recibir desde el AJAX la nueva información, es decir los valores actuales de las alarmas.

Para ello agregar un recuadro para los valores actuales de las alarmas que muestre: si hay movimiento, si esta conectada, si esta sonando la alarma de presencia, y si esta sonando la alarma de gas.

```
<fieldset>
 <legend>Valores Actuales de las alarmas</legend>
 <h4>Alarma de movimiento</h4>
 <br /><span class="valtitle">Estado:</span>
 <span id="ardPirMode" class="valvalue">?</span>
 <br /><span class="valtitle">Movimiento:</span>
 <span id="ardPirMove" class="valvalue">?</span>
 <br /><span class="valtitle">Sonando:</span>
 <span id="ardPirBeeping" class="valvalue">?</span>
 <h4>Alarma de gas</h4>
 <br /><span class="valtitle">Gas detectado:</span>
 <span id="ardGasBeeping" class="valvalue">?</span>
</fieldset>
```

6.4.4.2. Página web de estado en Javascript (index.js)

TODO 01 Leer los campos del buffer específico:

```
var vArdPirMode = miArray.substring(60,61);
//Alarma conectada [0-1]
var vArdPirMove = miArray.substring(61,62);
//Detectado movimiento [0-1]
var vArdPirBeeping = miArray.substring(62,63);
//Alarma de movimiento sonando [0-1]
var vArdGasBeeping = miArray.substring(63,64);
//Alarma de gas sonando [0-1]
```

TODO 02 Mostrar los campos del módulo específico, estados de la alarma de gas y movimiento y estado de los enchufes:

```
//Estado de la alarma: Conectada=1, desconectada =0
if(vArdPirConected==0)
{ document.getElementById("ardPirConected").innerHTML=
'Desconectada';}
else {
document.getElementById("ardPirConected").innerHTML=
'Conectada';}

// Detección de movimiento. No implica necesariamente
// disparo de la alarma.
if(vArdPirMove==0)
{document.getElementById("ardgetpirmove").innerHTML=
'Detectado movimento';}
else {document.getElementById("ardgetpirmove").innerHTML
='No hay movimiento' ;}
```

```
// Indicador de alama de movimiento sonando.
if (vArdPirBeeping==1)
{document.getElementById("ardPirFiring").innerHTML=
miSpanStarRed+'? ALARMA de Movimiento !'+miSpanEnd +
' sonando';}
else {$('#ardPirFiring').text('Alarma en Silencio');}

// Indicador de alama de gas sonando.
if (vArdGasBeeping==1)
{document.getElementById("ardgetgasalarm").innerHTML=
miSpanStarRed+'? ALARMA de Gas !'+miSpanEnd +
' sonando';}
else {$('#ardgetgasalarm').text('No se detecta gas');}
```

TODO 03 Añadir los mensajes a los relés en función de los nuevos modos de programación.

Los modos de programación se reciben del Arduino con un identificador numérico, y se muestran en la página como texto, para ello se utiliza esta matriz de arrays:

```
ar aMsg=["0  Manual OFF",
"1 Manual ON",
"2 Automático. Intervalo horario (Slot)",
"3 Automático. En función de temperatura
   e intervalo horario. (Calefactor)",
"4 Automático. En función de alarma de presencia",
"5 Automático. En función de  alarma de gas"]
```

6.4.5. Página web configuración de alarma y reloj

Los ficheros config.html y config.js son los que se modifican para permitir además de cambiar la fecha y hora del reloj (Modulo estándar) activar/desactivar la alarma de presencia PIR.

6.4.5.1. Página web de configuración en HTML (config.html)

En primer lugar programar la página config.html, agregando las etiquetas *span*, los campos de entrada de datos y botones de envío de comandos:

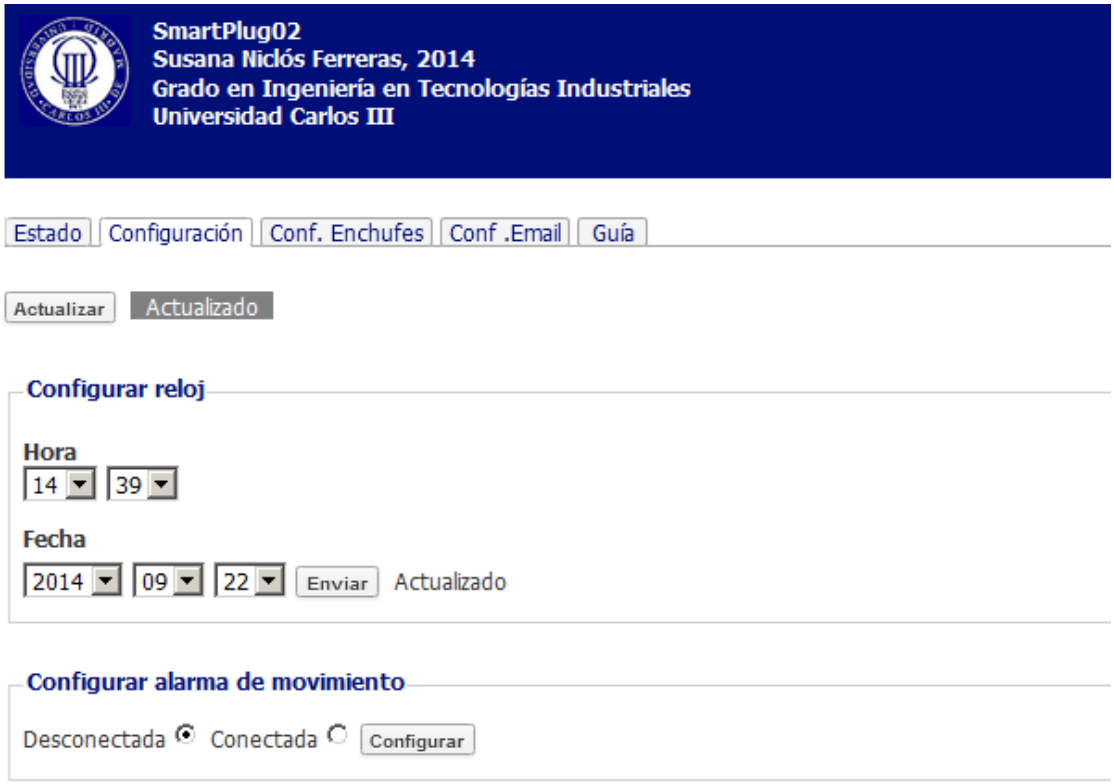


Figura 6.8: Campos adicionales del formulario config.html

Campos y botónes para activar la alarma:

- **Alarmoff:** Radio botón alarma desactivada.
- **Alarmon:** Radio botón alarma activada.
- **Botón enviar:** Llamar al función *fncSetAlarmFiringMode()*, JavaScript que mediante AJAX envía la configuración.
- **ArdgetAlarmFiringMode:** Campo informativo.

Código para agregar botones:

```
<fieldset>
<legend>Configurar alarma</legend>
Desconectada<input id="alarmoff" name="alarmsts" value="0"
title="OFF" type="radio" />
Conectada<input id="alarmon" name="alarmsts" value="1"
title="ON" type="radio" checked="checked" />

<input type="button" class="btnmini" value="Configurar"
 onclick="fncSetAlarmFiringMode();">
<span id="ardgetAlarmFiringMode" class="valvalue">.</span>
</fieldset>
```

6.4.5.2. Página web de configuración en Javascript (config.js).

TODO 01 Leer los campos del módulo específico.

```
//bPirAlarmFiringMode
var vArdPirMode = miArray.substring(60,61);
//bPirAlarmFiringMode
var vArdPirMove = miArray.substring(61,62);
//gm_bPirAlarmBeeping
var vArdPirBeeping = miArray.substring(62,63);
//digitalRead(PIN_GASLED)
var vArdGasBeeping = miArray.substring(63,64);
```

TODO 02 Mostrar los campos del módulo especifico alarmas. Se muestran en campos tipo radio-botón.

```
if (vArdPirConected=='1'){
  alarmon.checked = true;
  alarmoff.checked = false;
}
else{
  alarmon.checked = false;
  alarmoff.checked = true;}
}
```

6.4.6. Página web de configuración de la programación de enchufes

Los ficheros cfgplugs.html y cfgplugs.js son los que se modifican para permitir cambiar el intervalo de tiempo asignado a cada relé y su modo programación.

Solo es necesario agregar a los campos *select* los nuevos tipos de programación, en este caso el modo 2 se ha utilizado sin opción para la iluminación y se han agregado modos para poder activar los relés en función de la alarma de gas y de movimiento.

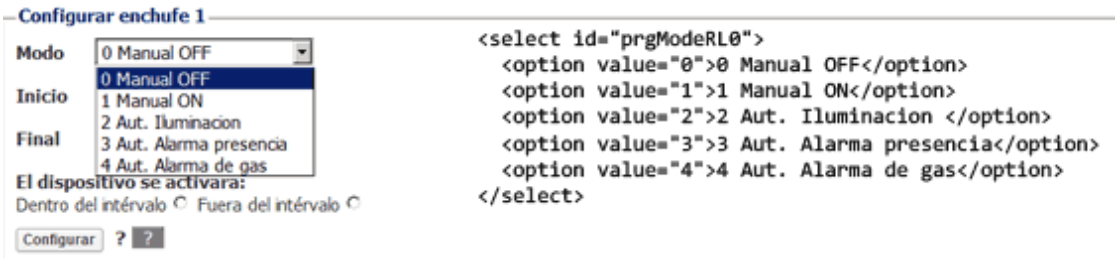


Figura 6.9: Página de programación de los enchufes.

6.4.6.1. Página de configuración de enchufes HTML (cfgplugs.html)

TODO 01 Agregar modos de programación para el relé 1, para ello localizar el campo *select* titulado *prgModeRL0* y agregar los cambios de modo de programación.

```
<legend>Configurar enchufe 2</legend>
<span class="valtitle2">Modo</span>
<select id="prgModeRL1">
  <select id="prgModeRL0" >
  <option value="0">0 Manual OFF</option>
  <option value="1">1 Manual ON</option>
  <option value="2">2 Aut. Iluminación </option>
  <option value="3">3 Aut. Alarma presencia</option>
  <option value="4">4 Aut. Alarma de gas</option>
  </select>
```

TODO 02, 03 y 04 Es equivalente al TODO1, pero para los otros relés. Repetir el proceso anterior respecto de los campos *select* titulados *prgModeRL1*, *prgModeRL2* y *prgModeRL3*.

6.4.6.2. Página configuración de enchufes en Javascript (cfgplugs.js)

No se necesita realizar cambios.

6.4.7. Página web de guía de usuario (guide.html)

Esta página tiene por objeto enseñar al usuario el manejo de la interfaz web, y configuración de conexiones WIFI. El programador de un nuevo módulo adaptado. la debe modificar para adaptase a sus necesidades tratando de conservar el estilo. Esta página no necesita una configuración Javascript especial.

Figura 6.10: Guía de usuario

6.4.8. Agregar nueva página para configurar datos de correo (cfgemail.html)

El objetivo de esta página es que el usuario pueda configurar la cuenta de correo que recibirá los emails de alerta.

Para crearla tomamos la página template.html y template.js y creamos una copia con el nombre cfgplugs.html y cfgplugs.js, luego se modifica su contenido para que permita cambiar los datos de conexión al servidor de correo (smtp).

Figura 6.11: Configurar cuenta de correo

6.4.8.1. Página para configurar datos del correo en HTML (cfgemail.html)

Se deben agregar los campos y botones siguientes:

Campo	Descripción
setsmtpuser	Identificación
setsmtpuserMsg	Mensaje de error
setsmtppwd	Contraseña
setsmtppwdMsg	Mensaje de error
setsmtpsrv	Nombre del servidor SMTP
setsmtpsrvMsg	Mensaje de error
setsmtpport	Puerto de conexión con el servidor
setsmtpportMsg	Mensaje de error
setemailto	Dirección de correo de destino del mensaje de alerta
setemailtoMsg	Mensaje de error
Botón enviar	Llama a la funciona fncSetConfigFile();
ardMsgBtn	Mensaje asociado al botón enviar

Para ello se debe modificar el código rellenando las etiquetas TODO.

TODO 01 Modificar nombre del fichero Javascript asociado.

```
<script type="text/javascript" src="js/cfgemail.js"></script>
```

TODO 02 Modificar el menú. Agregar opción para visualizar la página añadida.

```
<li class="active"><a href="cfgemail.html"
title="Configurar email">Conf .Email</a></li>
```

TODO 03 Modificar contenido. Añadir los campos necesarios, y mensajes de aviso de errores detectados, que rellenara mediante AJAX el la función JavaScript.

```
<fieldset>
<legend>Conexión al servidor de correo electrónico
</legend>

<label for="setsmtpuser">Usuario</label>
<input id="setsmtpuser" type="text" name="setsmtpuser"
value="hols" placeholder="user@domain.xxx"  required />
<span id="setsmtpuserMsg"  class="cred"></span>

<label for="setsmtppwd">Contraseña</label>
<input id="setsmtppwd" type="password" name="setsmtppwd"
placeholder="password" required/>
<span id="setsmtppwdMsg"  class="cred"></span>

<label for="setsmtpsrv">Servidor de correo</label>
<input id="setsmtpsrv" type="text" name="setsmtpsrv"
placeholder="smtp.gmail.com"  required/>
```

```
<span id="setsmtpsrvMsg"  class="cred"></span>

<label for="setsmtpport">Puerto</label>
<input id="setsmtpport" type="number"
onkeypress="return isNumberKey(event)" name="setsmtpport"
value=""placeholder="25 o 587 para gmail.com" required/>
<span id="setsmtpportMsg"  class="cred"></span>

<legend>Destinatarios del correo</legend>
<label for="setemailto">Email para</label>
<input  id= "setemailto" name="setemailto" type"email"
placeholder="email@domain.xxx" required/>
<span id="setemailtoMsg" class="cred"></span>

<br/><input type="submit" class="btnmini"
value="Configurar" onclick="fncSetConfigFile();">
<span id="ardMsgBtn" class="valvalue">.</span>
</fieldset>
```

6.4.8.2. Página para configurar datos del correo en Javascript (cfgemail.js)

Permite la lectura y modificación mediante Ajax del fichero *../www/py/-mail.conf*. Para almacenar los ficheros hace una llamada al fichero *cfgemail.py* pasandole como parámetros los datos a guardar.

Función	Descripción
fncOnLoad()	Se llama al cargar la página y llama a fncGetRefresh().
fncGetRefresh()	Llama a fncLoadingStart();
fncGetConfigFile()	Pide mediante AJAX el fichero el contenido del fichero /py/mail.conf y lo muestra en los campos.
fncSetConfigFile()	Guarda el fichero los datos del formulario en el fichero de configuración, llamando mediante AJAX al fichero cfgemail.py y pasando como parámetros los datos del formulario. Previo al envío, llama a la fncValidateFields() si esta función devuelve error, cancela la petición.
fncValidateFields()	Valida la coherencia de los datos del formulario, si hay errores los muestra en el formulario.
fncValidateEmail(IdObj.value)	Valida el patrón de escritura del email.
fncValidateTrim(IdObj)	Elimina espacios en blanco.
fncValidateisNumberKey(evt)	Evita que se tecleen letras en campo numérico, interceptando el teclado.

Capítulo 7

Planificación

Realizar una correcta planificación es de gran importancia, una buena estimación de tiempos puede ahorrar costes y marcar la diferencia entre satisfacer o no a un cliente.

Para una buena planificación se deben valorar bien los tiempos, y los recursos disponibles. Es importante tener en cuenta que sobrestimar el tiempo de todas las tareas no garantiza acabar en tiempo, sino que suele provocar retrasos en la fecha de fin del proyecto. Sin embargo sí es necesario añadir tiempo a las tareas que resulten críticas para asegurarse de que estas cumplen con la planificación.

Además, también es recomendable añadir un buffer de tiempo al final, para que en caso de que se retrase alguna tarea más de lo estipulado, aún se puedan cumplir con los plazos.

En este apartado se describirán dos planificaciones, la realizada al principio del proyecto, y la real.

7.1. Planificación inicial

Esta planificación es orientativa, es la primera estimación que se realizó. Para hacerla se ha dividido el proyecto en fases con tareas relacionadas y se han añadido los recursos, es decir personal trabajando en ellas.

Fase	Duración (h)	Personal
Análisis del problema	56	Desarrolladora: Susana Niclós Colabora Tutor: Pablo Zumel
Desarrollo módulo estándar	296	Desarrolladora: Susana Niclós
D. módulo ambiental	112	Desarrolladora: Susana Niclós
D. módulo control alarmas	112	Desarrolladora: Susana Niclós
Memoria	120	Desarrolladora : Susana Niclós Colabora Tutor: Pablo Zumel
Presentación	48	Desarrolladora : Susana Niclós
Total	632 horas	

Tabla 7.1: Estimación inicial de duración de las fases

Comenzando el 1 de Junio y utilizando como herramienta de estimación un diagrama de Gantt se espera finalizar el proyecto el 2 de Septiembre. Se considera que el desarrollador trabaja todos los días de la semana una jornada de 8 horas y que tiene formación de ingeniero junior. Es importante tener en cuenta que si este proyecto lo estuviera realizando un ingeniero senior los tiempos serían muy inferiores.

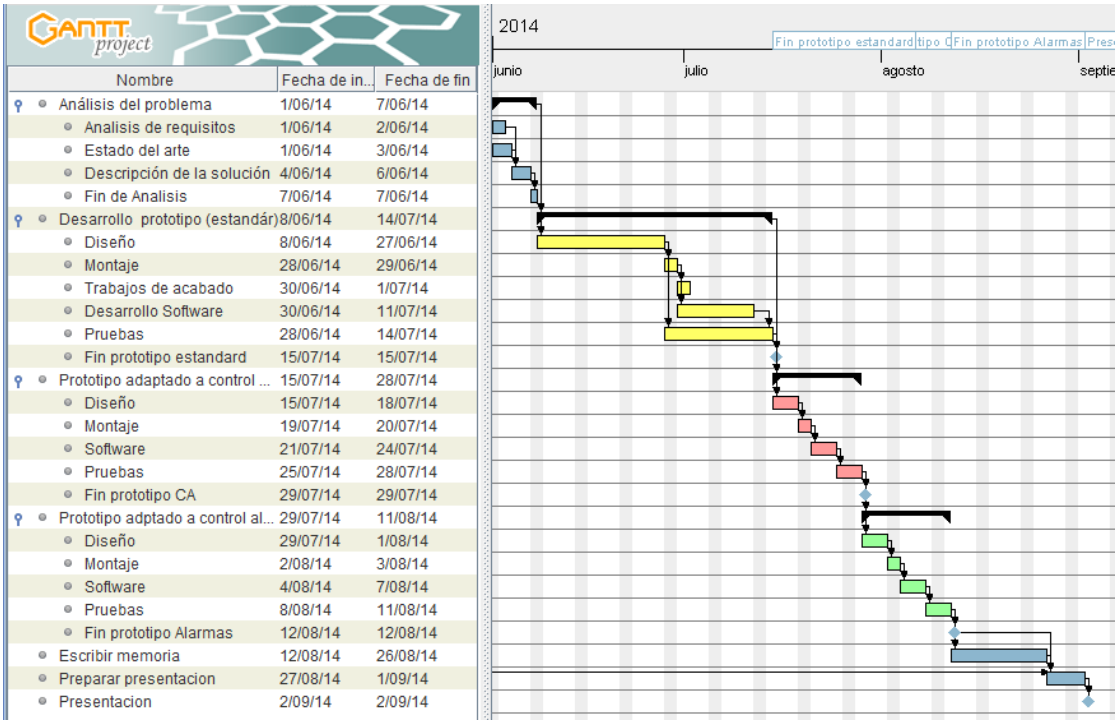


Figura 7.1: Diagrama de Gantt: Planificación Inicial

7.2. Planificación final

La planificación final es la real, tiene en cuenta los tiempos que realmente han sido necesarios. En la siguiente tabla se encuentran las fases con sus tiempos reales y el personal necesario para ejecutarlas.

Fase	Duración (h)	Personal
Análisis del problema	56	Desarrolladora: Susana Niclós Colabora Tutor: Pablo Zumel
Desarrollo módulo estándar	336	Desarrolladora: Susana Niclós
D. módulo ambiental	120	Desarrolladora: Susana Niclós
D. módulo control alarmas	128	Desarrolladora: Susana Niclós
Memoria	128	Desarrolladora: Susana Niclós Colabora Tutor: Pablo Zumel
Presentación	48	Desarrolladora: Susana Niclós
Total	816 horas	

Tabla 7.2: Estimación final de duración de las fases

En la planificación inicial se esperaba finalizar el proyecto el 2 de Septiembre, sin embargo realmente el proyecto ha finalizado el 11 de Septiembre, esto es un retraso de solo 9 días por lo que no resulta muy grave. A continuación se encuentra el diagrama de Gantt con los tiempos reales:

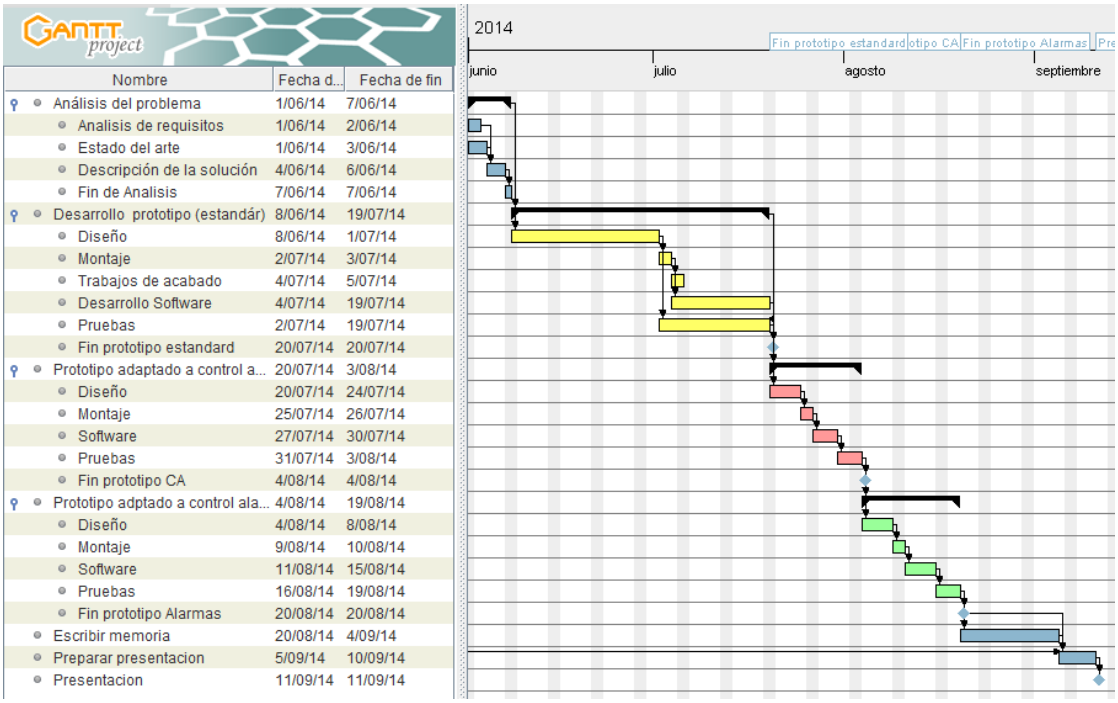


Figura 7.2: Diagrama de Gantt: Planificación Final

Capítulo 8

Presupuesto

8.1. Costes de desarrollo

Los costes de desarrollo se dividen en coste de materiales, coste personal y coste de utilización de herramientas. En cada apartado aparecerán dos tablas, la primera con los costes estimados en un principio y la segunda con los reales. Todos los precios vienen indicados en euros salvo que se indique lo contrario.

8.1.1. Coste de material

En este apartado se describen los costes de los materiales que han sido necesarios para desarrollar el Smartplug, algunos materiales se encuentran por duplicado ya que se optó por construir dos modelos distintos para verificar que era posible desarrollar un producto de gran adaptabilidad.

Coste estimado de material

Producto	Descripción	Cantidad	Coste unidad (€)	C. total (€)
Arduino Yún	Placa microcontroladora	2	65	130
Ywrrobot 4 relay board	Placa de relés	2	6	12
DS1307	Reloj de tiempo real	2	3	6
Fuente de alimentación	220V AC a 5V DC	2	6	12
Cables Jumpers	Cables para placa protoboard	2 lotes	2	4
Protoboard	Placa para desarrollo	2	8	16
Cable	Cable varios diámetros	10 m	0.5 €/m	5
PIR	Sensor de presencia	1	5	5
DHT22	Sensor de temperatura	1	6	6
MQ5	Sensor de gas	1	6	6
Carcasa	Piezas impresas en 3d	2 lotes	27 €/lote	54
Soldador	Nota 1	1	50	50
Regleta 4 enchufes	Marca Aki 9667776	2	9	18
Total				324

Tabla 8.1: Coste estimado de material en el proceso de desarrollo

Nota: aunque técnicamente el soldador constituye una herramienta, por su bajo precio se ha incluido en los materiales.

Coste real de material

Producto	Descripción	Cantidad	Coste unidad (€)	Coste total(€)
Arduino Yún	Placa microcontroladora	2	65	130
Ywrrobot 4 relay board	Placa de relés	2	6	12
DS1307	Reloj de tiempo real	2	3	6
Fuente de alimentación	220V AC a 5V DC	5	6	30
Cables Jumpers	Cables para placa protoboard	2 lotes	2	4
LM7805	Regulador de Tensión	2	1.5	3
Protoboard	Placa para desarrollo	2	8	16
Cable	Cable varios diámetros	12 m	0.5 €/m	6
PIR	Sensor de presencia	1	5	5
DHT22	Sensor de temperatura	3	6	18
MQ5	Sensor de gas	1	6	6
Carcasa	Piezas impresas en 3d	2 lotes	27 €/lote	54
Soldador	Nota 1	1	50	50
Regleta 4 enchufes	Marca Aki 9667776	4	9	36
Total				376

Tabla 8.2: Coste real de material en el proceso de desarrollo

Resultado de la comparación

Como se puede ver los costes reales son mayores que los estimados. Esto es debido a los componentes que fue necesario comprar para reemplazar aquellos que habían ido fallando durante el proceso de desarrollo, y a los componentes adicionales que fue sido necesario adquirir, y que no se habían previsto.

Sin embargo como se trata de componentes de bajo precio la diferencia tampoco resulta muy grande, por lo que se puede concluir que la estimación inicial resulta bastante adecuada.

8.1.2. Coste de herramientas

Para la elaboración de este proyecto se preveían necesarias diversas herramientas, ya que se van a utilizar durante un periodo muy corto y después pueden ser utilizadas para otras funciones se ha puesto el precio de utilización.

Coste estimado de herramientas

Producto	Precio (€)	Tiempo de utilización	Coste para el proyecto (€)
Toshiba Tecra	500	3 meses	250
Desarrollador Catia	20 €/hora	40 horas	800
Total			1050

Tabla 8.3: Estimación inicial de coste de herramientas

Coste real de herramientas

Producto	Precio (€)	Tiempo de utilización	Coste para el proyecto (€)
Toshiba Tecra	500	3 meses	250
Desarrollador Catia	20 €/hora	60 horas	1200
Total			1450

Tabla 8.4: Coste real de herramientas

Resultado de la comparación

El tiempo de desarrollo de la carcasa fue algo mayor de lo esperado, pero como en el caso anterior la estimación fue bastante adecuada.

8.1.3. Coste personal

El coste personal utiliza los resultados de las dos planificaciones con una estimación aproximada de los sueldos actuales.

Coste personal estimado

Nombre	Categoría	Salario (€/Hora)	Horas Totales	Coste total (€)
Susana Niclós	Ingeniero Junior	15	632	9480
Total				9480

Tabla 8.5: Coste personal estimado

Coste personal real

Nombre	Categoría	Salario (€/Hora)	Horas Totales	Coste total (€)
Susana Niclós	Ingeniero Junior	15	816	12240
Total				12240

Tabla 8.6: Coste personal real

Resultado de la comparación

En este caso la diferencia es mucho más significativa, esto es debido a que la planificación no ha sido exacta, y ha afectado a los costes personales.

Por esto es tan importante realizar una buena planificación, ya que de ella se obtiene la estimación de costes humanos y estos suelen ser los mas grandes en este tipo de proyectos.

8.1.4. Coste total de desarrollo del proyecto

En la siguiente tabla se compararan los costes totales estimados frente a los reales.

Objeto del Coste	Precio Estimado (€)	Precio Real (€)
Materiales empleados	324	376
Herramientas utilizadas	1050	1450
Coste Personal	9480	12240
Total	10854	14066

Tabla 8.7: Costes totales estimados y reales

Resultado de la comparación

Finalmente se puede ver que los costes reales son en torno a 4000 €mayores de lo esperado, esto no es demasiado para el tamaño del proyecto, pero se debe revisar la planificación para realizar una mejor estimación en la segunda fase.

8.2. Costes de fabricación

En esta tabla se estima los costes de fabricación una unidad Smartplug.

Producto	Descripción	Cantidad	Coste unidad (€)	C. total (€)
Arduino Yún	Placa microcontroladora	1	65	65
Ywrrobot 4 relay board	Placa de relés	1	6	6
DS1307	Reloj de tiempo real	1	3	3
Fuente de alimentación	220V AC a 5V DC	1	6	6
Cable	Cable varios diámetros	2 m	0.5 €/m	1
Carcasa	Piezas impresas en 3D	1 lotes	27 €/lote	27
Sensores	Precio medio sensor	2	6	12
Montaje	Técnico encargado de montaje	14€/hora	0.5 hora	7
Regleta 4 enchufes	Marca Aki 9667776	1	9	9
Total				136

Tabla 8.8: Coste de fabricación de una unidad de Smartplug

Capítulo 9

Conclusiones

9.1. Conclusión personal

Tras haber estado trabajando durante un largo tiempo en este proyecto considero que el resultado ha sido muy satisfactorio. Se ha conseguido encontrar una solución al problema inicial y llevarla a cabo.

Creo que este proyecto me ha aportado mucho ya que he aprendido a utilizar distintos lenguajes de programación y software durante su desarrollo. Además he aplicado los conocimientos adquiridos durante mi grado, especialmente de electrónica y organización.

Me parece importante mencionar que desde que se empezó este proyecto multitud de empresas han desarrollado dispositivos similares, pero considero que el Smartplug sigue ofreciendo características competitivas que le diferencian y que le hacen mejor.

Finalmente, creo que este proyecto tiene un gran potencial porque aunque la idea es vender Smartplug ya montados y listos para usar, también se quiere hacer un kit para desarrolladores, y crear una plataforma abierta basada en el módulo estándar sobre la que se desarrollen proyectos.

9.2. Estado actual

Actualmente se ha logrado cumplir con el objetivo de este proyecto llevando a cabo la construcción de un prototipo totalmente funcional del módulo estándar (Smartplug), y dos prototipos de módulos especializados: el módulo de control ambiental (Smartplug01) y el módulo de alarma (Smartplug02). Además se ha verificado la compatibilidad de estos.

También se ha creado un entorno web en HTML5, fácil de usar y visualizable desde cualquier navegador compatible, y se ha diseñado una carcasa que contiene toda la electrónica y que permite que este dispositivo sea seguro.

En la siguiente tabla se encuentran los requisitos que se plantearon desde un principio, y el estado en que se encuentran actualmente.

Requisito	Cumplido	No cumplido
Diseño modular	✓	
Producto utilizable sin necesidad de realizar instalaciones	✓	
Compatibilidad con cualquier dispositivo que se enchufe	✓	
Personalizable	✓	
Módulo estándar preparado para usado como kit de desarrollollo	✓	
Control de cuatro enchufes con programación horaria	✓	
Control de cuatro enchufes con distintos sensores	✓	
Entorno web sencillo sin necesidad de instalación	✓	
Lectura del estado de los sensores a través del entorno web	✓	
Programación de los relés utilizando el entorno web	✓	
Avisos a través del correo	✓	
Diseño de una carcasa adaptable	✓	
Elaboración de un prototipo funcional del módulo estándar	✓	
Elaboración de un prototipo funcional de módulo especializado	✓	
Integración de ambos prototipos	✓	
Seguridad en comunicaciones		✓
Bajo precio		✓
Pequeño tamaño		✓
Creación de una página para desarrolladores		✓

Tabla 9.1: Requisitos planteados

Nota: los objetivos no cumplidos serán realizados en la segunda fase.

Problemas resueltos durante el desarrollo

Considero necesario mencionar los principales problemas a los que me he enfrentado durante el desarrollo de este proyecto y las soluciones que he encontrado. Resolver e identificar estos fallos ha llevado una gran cantidad de tiempo, y creo que puede resultar útil describirlos para futuros proyectos.

Principalmente estos problemas han sido debidos a que se está trabajando con una primera versión de Arduino Yún, y utilizando un entorno de desarrollo Beta,

aunque también existen a otros factores.

Uno de los mayores problemas a los que me he enfrentado en este proyecto es la falta de memoria en el microprocesador ATmega32U4, que ha obligado a desestructurar el código, a realizar tareas en el otro microprocesador y a eliminar algunas funciones. Parte de estos problemas de falta de memoria eran causados por la primera versión del compilador de Arduino Yún, en la nueva versión disponible desde Agosto de 2014 se logra un uso más eficiente de la memoria, y el mismo código ocupa menos tamaño.

El segundo problema con Arduino han sido los fallos causados por la librería *Bridge* encargada del envío y recepción de datos entre los dos microprocesadores, estos ya han sido reportados al fabricante y están pendientes de solución definitiva en la segunda versión. Los errores de esta librería provocaban fallos en las comunicaciones, pérdida de información y obtención valores aleatorios. Actualmente el fabricante proporciona soluciones provisionales que se encuentran descritas con detalle en el capítulo 4 de esta memoria.

Otro gran problema ha sido la imposibilidad de alimentar Arduino utilizando los puertos Vin y GND. Desde un primer momento todo el diseño estaba orientado a realizar la alimentación por estos puertos, sin embargo esto no ha sido posible ya que incluso utilizando un regulador Arduino experimentaba un comportamiento inestable. Finalmente fue necesario realizar modificaciones sobre todo en el diseño de la carcasa para poder alimentar el arduino a través del puerto microUSB.

También cabe mencionar que el montaje se ha llevado mucho tiempo debido a que al estar usando componentes independientes se requería mucho cableado, y en ocasiones se producían problemas (cortocircuitos, malos contactos...). Esto se resolverá en la siguiente fase del proyecto cuando se diseñe una placa para toda la electrónica común y se reduzca significativamente los cables necesarios.

9.3. Mejoras futuras

Este proyecto se halla en fase de desarrollo por lo que aún existen una gran cantidad de mejoras que deben ser aplicadas antes de comercializar el producto. Principalmente estas tienen que ver con la reducción de costes, de tamaño, aumento de la seguridad, aunque tras realizar estas mejoras se buscará ampliar la gama de productos disponibles.

9.3.1. Desarrollo de una placa integrada

Como ya se ha mencionado en las distintas partes de esta memoria, el siguiente paso de este proyecto será desarrollar una placa microcontroladora adaptada específicamente al Smartplug, que permitirá reducir costes y optimizar el espacio.

Esta placa incluirá todo el hardware presente en el módulo estándar, es decir: un microprocesador, el reloj de tiempo real, 20 puertos GPIO con protecciones contra sobreintensidades, y 6 analógicos. Además encajará a la perfección con una segunda

placa que contendrá los relés y la fuente de alimentación, esto permitirá adaptarse de forma más sencilla a países con distintas normativas eléctricas. También, para reducir el tiempo de conexión y dar mayor solidez al producto se diseñaran distintas placas de sensores para los módulos especializados que encajaran directamente sobre los puertos GPIO y analógicos, evitando así tener que usar cables.

Esta placa será la base para los desarrolladores, que podrán diseñar sus propios módulos especializados en vez de utilizar los fabricados y conectarlos directamente al módulo estándar. Así, ya partirán de un módulo capaz de controlar cuatro enchufes utilizando la red WIFI, y desde ahí podrán crear sus proyectos.

Ya que los principales problemas de este proyecto han sido la falta de memoria en el microcontrolador ATmega32U4 y los problemas de comunicación entre los dos microprocesadores, esta placa solo incluirá un microprocesador de mayor potencia encargado de todas las funciones, es decir, leer y escribir en los puertos GPIO, actuar como servidor y disponer de conexión WIFI.

Principalmente la placa desarrollada será una combinación de Arduino y Raspberry Pi B+, pero se eliminará toda la electrónica que resulte innecesaria y se incluirá la específica del proyecto.

9.3.2. Desarrollo de otros módulos especializados

En todo momento se ha buscado desarrollar un producto versátil, en el que modificando solo una pieza se pudiera obtener un producto con funcionalidades diferentes.

Por ello desde el principio se optó por un desarrollo modular, en el que sobre un módulo estándar se pueden incorporar distintos módulos adicionales para incrementar las funcionalidades. El usuario podrá escoger entre comprar módulos ya fabricados o diseñar los suyos propios.

Actualmente solo se han creado dos módulos especializados, el de control ambiental y el sistema de alarma, sin embargo se planea ampliar las opciones disponibles. Se quiere incrementar la oferta de módulos fabricados con funcionalidades concretas pero también dar la posibilidad al ciente de escoger sus propios sensores entre los disponibles.

Para incorporar estos cambios es necesario partir de una placa con mayor memoria que permita desarrollar la programación de cada sensor en clases independientes, así al construir el dispositivo solo es necesario agregar el código pertinente.

9.3.3. Diseño total de la carcasa

En este primer prototipo para la pieza de la clavija y para la de los enchufes se ha utilizado una regleta comercial, por lo que todo el diseño se ha visto influenciado por el tamaño y forma de esta.

Sin embargo, para el producto final lo adecuado sería fabricar todas las piezas

de la carcasa, adaptándolas a la electrónica que se va a utilizar, y optimizando así el espacio.

El modo de fabricación ya no sería utilizando una impresora 3D, ya que este método solo resulta adecuado para realizar prototipos, sino que se buscaría un método más económico para la producción en masa, en el que las piezas tengan mayor resistencia y solidez, por ejemplo inyección de plástico.

La inyección de plástico resulta perfecta ya que permite elaborar piezas complejas sin necesidad casi de acabado final, a bajo coste para altos niveles de producción.

9.3.4. Creación de una plataforma para desarrolladores

Como ya se ha mencionado antes, se planea crear una plataforma abierta para desarrolladores basada en el módulo estándar.

El objetivo de esta plataforma es facilitar el desarrollo y difusión de nuevos proyectos, favorecer al intercambio de conocimientos y la resolución de dudas.

Este proyecto tiene un gran potencial como kit para desarrolladores, y con un buen impulso inicial se cree que esta plataforma será capaz de automantenerse.

9.3.5. Mejoras en la seguridad de software

Actualmente este proyecto es totalmente funcional, sin embargo todavía carece de las medidas de seguridad en el software necesarias para hacerlo comercial. Estas medidas son muy necesarias ya que este dispositivo tendrá acceso a electrodomésticos, luces y otros aparatos eléctricos... y puede ser peligroso que alguien no autorizado manipule la programación.

El primer nivel de seguridad será añadir una página de acceso previa a los menús de configuración, en la que será necesario introducir un usuario y una contraseña. Estos vendrán asignados por defecto al comprar un Smartplug, aunque podrán ser cambiados desde la página web.

Además, con el objetivo de mejorar la seguridad en las comunicaciones, se planea añadir un sistema de cifrado de datos. Estos sistemas son sencillos de desarrollar, aunque necesitan una placa con mayor memoria.

Bibliografía

[1] Real Academia Española, (2014). RAE. Consultada 6 de Septiembre 2014, en http://lema.rae.es/drae/?val=domoticabook.

[2] DomoDesk, (2014). Domodesk. Consultada 6 de Septiembre, en http://www.domodesk.com/que-es-domotica.

[3] SmartHomes, (2014). SmartHomes. Consultada 3 de Septiembre, en http://www.smart-homes.nl/Domotica.aspx?lang=en-US.

[4] Construcmática, (2014). Construcpedia. Consultada 2 de Septiembre, en http://www.construmatica.com/construpedia/Dom%C3%B3tica.

[5] Arqcompus domótica (2014).Arq domótica. Consultada 4 de Septiembre, en http://arqcompus-domotica.blogspot.com.es/

[6] Smarthome store (2014).Smarthome. Consultada 4 de Septiembre, en http://www.smarthome.com/sc-what-is-x10

[7] European Committee for Electrotechnical Standardization (2014). Cenelec. Consultada 4 de Septiembre, en http://www.cenelec.eu/.

[8] Biodom (2014). Biodom. Consultada 4 de Septiembre, en http://bioingenieria.es/BioDom/BioDom.htm

[9] Serconint automatización, (2014).Serconint. Consultada 4 de Septiembre, en http://www.serconint.com/sistemasprotocolos.php

[10] Janko Roettgers. (2014). BusinessWeek . Consultada 5 de Septiembre, en http://www.businessweek.com/articles/2013-05-08/time-for-googles-android-at-home-to-make-a-new-splash

[11] Apple developer. (2014). Apple . Consultada 5 de Septiembre, en https://developer.apple.com/homekit/

[12] D-link, (2014). D-Link. Consultada 7 de Septiembre, en http://www.dlink.com/us/en/home-solutions/share/network-attached-storage/dsp-w215.

[13] Belkin, (2014). Belkin. Consultada 7 de Septiembre, en http://www.belkin.com/us/F7C029-Belkin/p/P-F7C029/.

[14] EasySmart,(2014). EasySmart. Consultada 7 de Septiembre, en http://easysmart.com/computer-wifi-controlled-electric-socket.

[15] Arduino, (2014). Arduino Yún board. Consultada 4 de Junio, en http://arduino.cc/en/Main/ArduinoBoardYun.

[16] Raspberrypi, (2014). Raspberrypi board B. Consultada 4 de Junio, en http://www.raspberrypi.org/products/model-b/.

[17] Beagleboard, (2014). Beagleboard. Consultada 4 de Junio, http://www.electronicafacil.net/tutoriales/El-rele.php.

[18] Catia, (2002-2014) Dassault Systèmes, Software de diseño 3D Catia, en http://www.3ds.com/es/productos-y-servicios/catia/

[19] Electrónica fácil, (2014). Electrónica fácil. Consultada 8 de agosto, en http://beagleboard.org/bone.

[20] Songle (2013). Songle datasheet ISO9002. Consultada 8 de agosto, en http://www.datasheet-pdf.com/datasheet/Songle/720556/SRD-05VDC-SL-C.pdf.html.

[21] Hanwey electronic CO.,LTD, (2014). Technical data MQ-5 gas sensor. Consultada 6 de junio 2014, en http://www.seeedstudio.com/depot/datasheet/MQ-5.pdf

[22] Cytron Technologies, (2007). User's Manual V1.1 December 2007 PIR Sensor. Consultada 6 de junio 2014, en http://www.cytron.com.my/datasheet/sensor/PIR_UserManualv1.pdf

[23] W3C (MIT, ERCIM, Keio, Beihang),(2014). HTML 5 Recomendations. Consultada 1 de agosto 2014, en http://www.w3.org/TR/html5/

[24] Adrafruit,(2014). Adafruit. Consultada 3 de agosto 2014, en http://www.adafruit.com/products/385

[25] Arduino (2014). Arduino 1.5.7 BETA (with support for Arduino Yún and Arduino Due boards), en http://arduino.cc/en/pmwiki.php?n=main/software

[26] Putty Org. (2014). Putty. Consultada 8 de Agosto 2014, en http://www.putty.org

[27] WinSCP.(2014). WinSCP. Consultada 10 de Agosto 2014, en http://winscp.net/eng/docs/lang:es

[28] Arduino. (2014). Getting Started with Yún. Consultada 15 de Agosto 2014, en http://arduino.cc/en/Guide/ArduinoLeonardoMicro?from=Guide.ArduinoLeonardo

Apéndice A

Putty

Putty es un cliente SSH, Telnet, rlogin, y TCP raw con licencia libre [26]. En este proyecto se ha utilizado este programa para acceder a los archivos del microprocesador de linux y de la microSD.

Guía de usuario

Paso 1 Descarga e instalación: Es posible descargarlo para varias plataformas (Windows,Mac, Unix) de su página oficial de manera gratuita.

`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`

Paso 2 Introducir datos en el menú general: Al abrir el programa se accede a un menú como el de la figura A1. Hay que introducir la dirección IP de su Arduino en la casilla Host Name(or IP adress) y hacer click en Open.

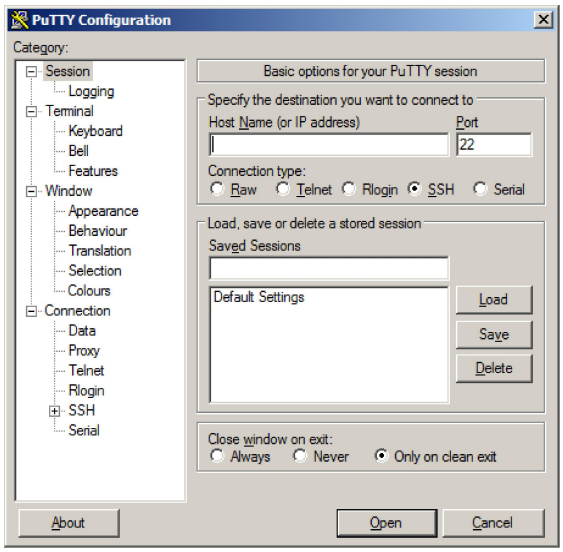


Figura A.1: Menú general Putty (Paso 2)

Nota: Para conocer la dirección del dispositivo utilizar el programa Ipscan23 (consultar apéndice E).

Paso 3 Introducir usuario y contraseña: Si todo ha ido correctamente aparecerá una ventana de comando como la de la figura A.2 solicitado usuario y contraseña.

usuario: root **contraseña:** arduino

Figura A.2: Ventana de comando Putty (Paso 3)

Apéndice B

WinSCP

WinSCP es un programa que utiliza el protocolo SFTP para conectarse a un servidor SSH. Así permite la transferencia de archivos con clientes FTP utilizando un sencillo entorno gráfico [27].

Este programa puede ser empleado para modificar los ficheros de configuración internos del procesador linux de Arduino. Aunque también sería posible realizar estas modificaciones utilizando el Putty, esto requeriría muchos comandos por resulta mucho más sencillo hacerlo con el WinSCP.

Guía de usuario

Paso 1 Instalación de la librería: Antes de comenzar a utilizar este programa es necesario instalar en nuestro dispositivo las librerías para la comunicación SFTP. Para ello debemos conectarnos al Arduino utilizando Putty e introducir las siguientes sentencias:

```
opkg update
opkg find openssh-sftp-server:
opkg install openssh-sftp-server:
```

Este paso solo es necesario ejecutarlo la primera vez, aunque en caso de fallo o reseteo del Arduino puede ser necesario repetirlo, para ello debemos sustituir la ultima sentencia por la siguiente:

```
opkg install --force-reinstall openssh-sftp-server
```

Paso 2 Descarga e instalación: Es recomendable descargar el programa en el siguiente enlace:

`http://winscp.net/eng/download.php`

Paso 3 Menú general: Al abrir el programa aparece el menú general (Figura B.1) debemos pulsar en la izquierda sobre *Nuevo Sitio* e introducir los datos de acceso:

- **Ip o Nombre del Servidos:** IP de Arduino
- **Usuario:** root
- **Contraseña:** arduino

En caso de no conocer a IP actual del Arduino utilizar el Ipscan23 (consultar apéndice E). Tras introducir todos los datos solo es necesario hacer click en conectar.

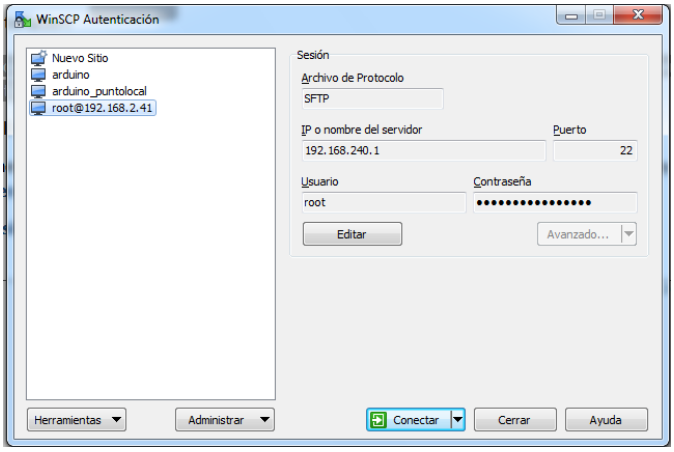


Figura B.1: Menú General (Paso 3)

Paso 4 Descargar y editar: Si se ha logrado conectar aparecerá una imagen como la de la figura B.2 con dos ventanas con carpetas, a la izquierda su Pc, y a la derecha el Arduino. Para descargar o subir archivos solo es necesario arrastrarlas del Pc al Arduino y viceversa.

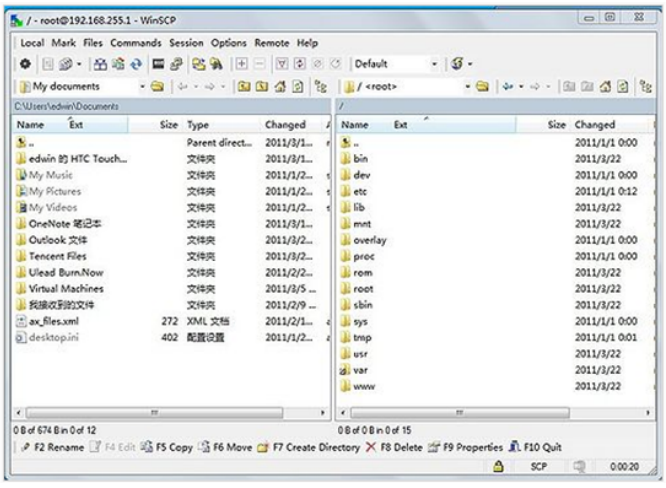


Figura B.2: Conexión con Arduino realizada (Paso 4)

Apéndice C

Panel de Control Web Arduino

El panel de control web de Arduino viene instalado por defecto, cuenta con una serie de menús para administrar los parámetros más generales del linux, y tiene un acceso a Luci (entorno gestor del microprocesador linux, consultar apéndice D).

Desde este panel es posible configurar parámetros como el nombre de usuario y contraseña, o los datos de la red. Además cuenta con una herramienta de diagnóstico sobre el estado de las conexiones WIFI y Ethernet [28]. Sin embargo, esta herramienta solo permite configurar cosas muy generales y si se desea acceder a configuraciones más especificas del sistema es necesario acceder al Luci.

Guía de acceso al Panel de Control Web

Opción A: Guía para Arduinos configurados como punto de acceso

Si es la primera vez que utilizas tu Arduino y no has cambiado la configuración de redes o si has reseteado los parametros WIFI de Arduino debes seguir estos pasos.

Paso 1 Conectarse a la red Arduino: tras enchufar el Arduino, al cabo de unos segundos en el administrador de redes del PC aparecerá una red con nombre: *ArduinoYun-XXXXXXXXXXXX* , debemos conectarnos a ella.

Paso 2 Acceder al panel web : Una vez conectados, utilizando un navegador debemos entrar en la dirección `http://192.168.240.1` Así aparecerá una pagina como la de la figura C.1

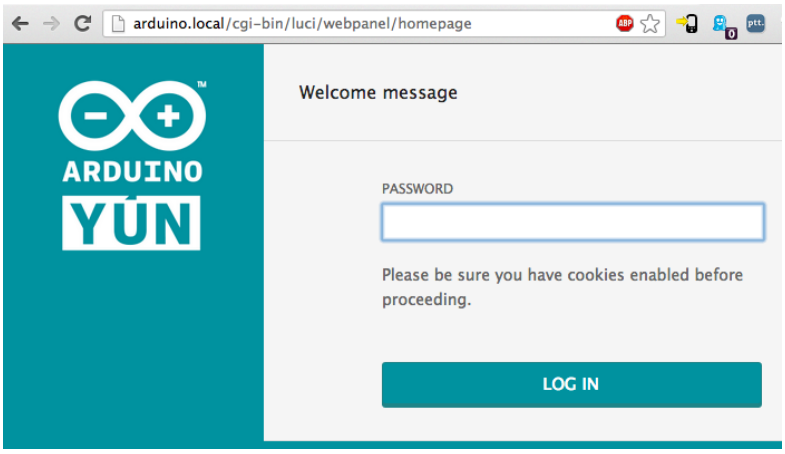


Figura C.1: Acceso al panel web (Paso 2)

Paso 3 Introducir Contraseña: Al introducir la contraseña y hacer click en *LOG IN* entrara en el panel de configuración web y se tendrá acceso a todos los menús de administración del sistema.

Contraseña: Arduino

Opción B: Guía para Arduinos configurados como clientes Web

Si se ha realizado configuración inicial descrita en el apartado 4.8.2 es decir, si se a configurado el Arduino como cliente web, deben seguirse los siguientes pasos:

Paso 1 Identificar dirección Ip del Arduino: Utilizando un escaneador de redes como el Ipscan23 (consultar apéndice E) averiguar la dirección Ip actual de Arduino.

Paso 2 Acceder al panel web: Utilizando un navegador web conectarse a la página: `http://<Ip-Arduino>`

A partir de ahí seguir desde el Paso 3 de la Opción A: Guía para Arduinos configurados como punto de acceso.

Menús de administración del sistema

Herramienta de diagnóstico: Es el primer menú que aparece al introducir la contraseña. La herramienta de diagnóstico (figura C.2) proporciona información sobre el estado de las conexiones WIFI y Ethernet, la cantidad de MB transmitidos y recibidos y las direcciones Ip que poseen.

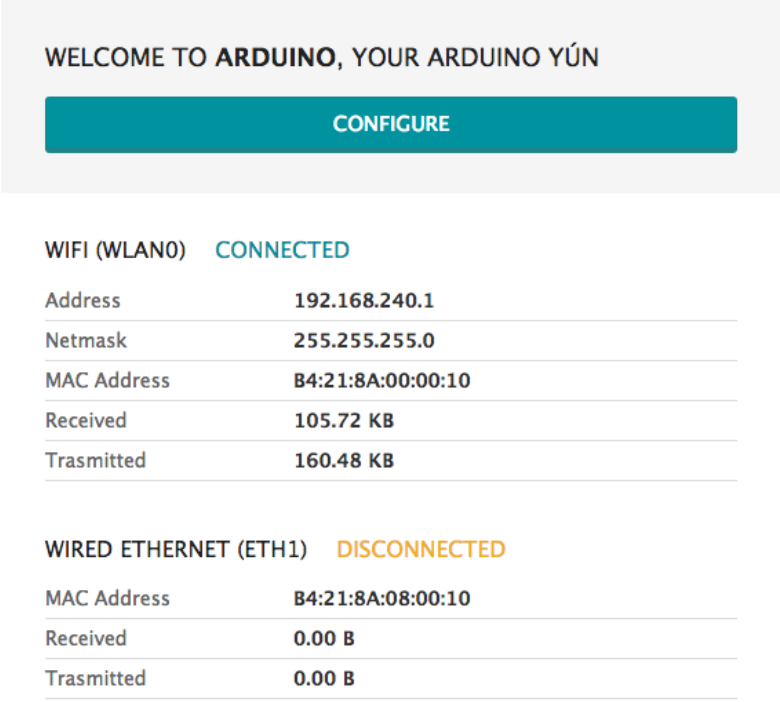


Figura C.2: Herramienta de diagnóstico de conexiones

Menú de configuración: Para acceder a este menú se debe hacer click en *CONFIGURE* desde la página de la herramienta de diagnóstico. Este menú (figura C.4) se divide en dos partes :

1. **Yún board configuration:** Permite modificar el nombre de usuario, la contraseña y establecer nombre de nuestro dispositivo.

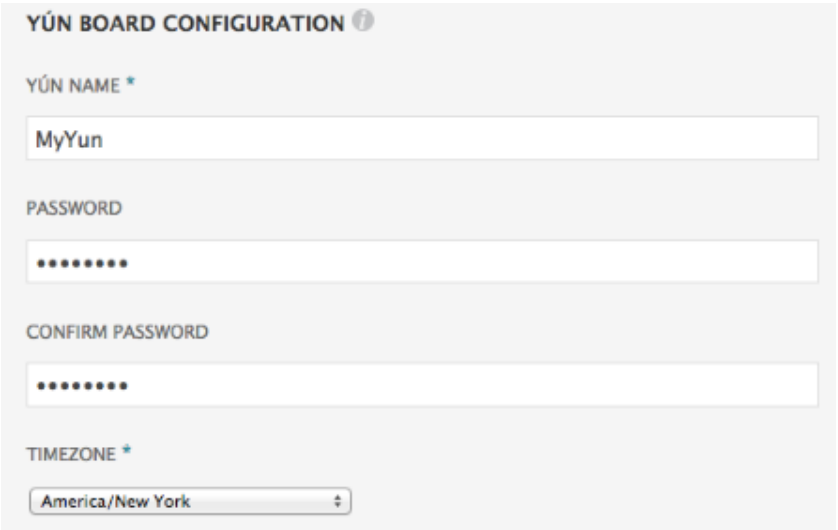


Figura C.3: Menú de Configuración Yún Board

2. **Wireless parameters:** Permite configurar los datos de la red a la que se desea que se conecte el Arduino. Cabe mencionar que por defecto Arduino Yún funciona como punto de acceso, pero que al introducir una red en este menú pasa a funcionar como cliente dentro de una red.

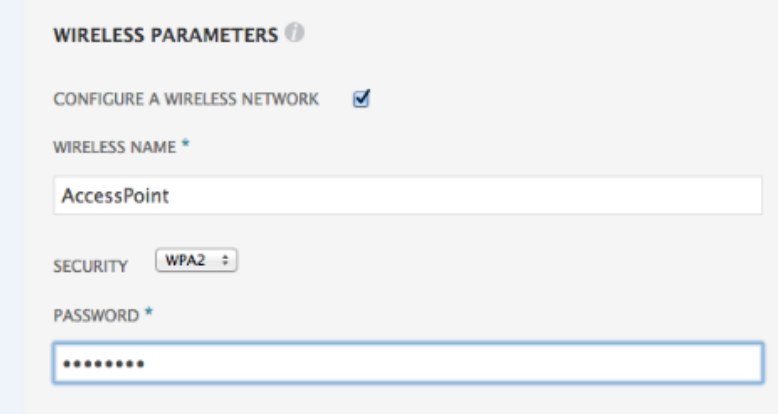


Figura C.4: Menú de Configuración Wireless Parameters

Apéndice D

Luci

Luci es un interfaz que viene instalado por defecto en el microprocesador de linux AR9331 y permite gestionar la parte linux sin necesidad de utilizar la consola. Utilizando Luci es posible configurar redes, administrar las procesos activos, analizar el uso de la memoria, y otros procesos relacionados con los parametros de configuración de linux [28].

Luci tiene disponibles una gran cantidad de menús administrativos y de configuración sin embargo en este proyecto solo ha utilizado el menú Startup .

Guía de acceso a Luci

Para acceder a Luci debemos entrar en el panel control web y desde el menú de configuración hacer click sobre *advance configuration panel (luci)* figura D.1.

Para más información sobre como acceder al menú de configuración consultar: La guía de acceso al Panel de Control Web apéndice C

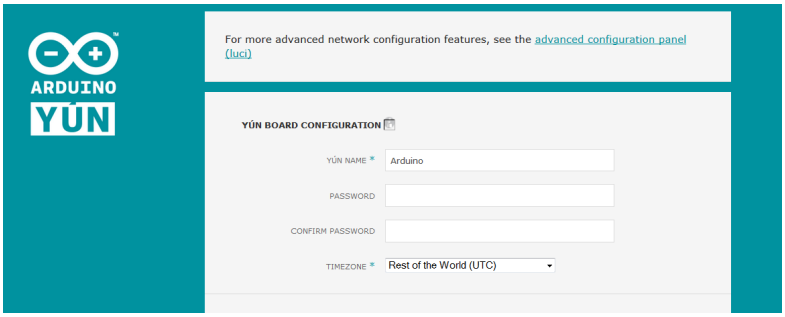


Figura D.1: Acceso al Luci

Guía de acceso al menú Startup

El menú Startup se encuentra en el menú superior dentro de las opciones de la categoría System. Con el Startup (figura D.2) es posible habilitar y deshabilitar

los procesos que se activan al arrancar el microprocesador linux e incluso pararlos o resetearlos.

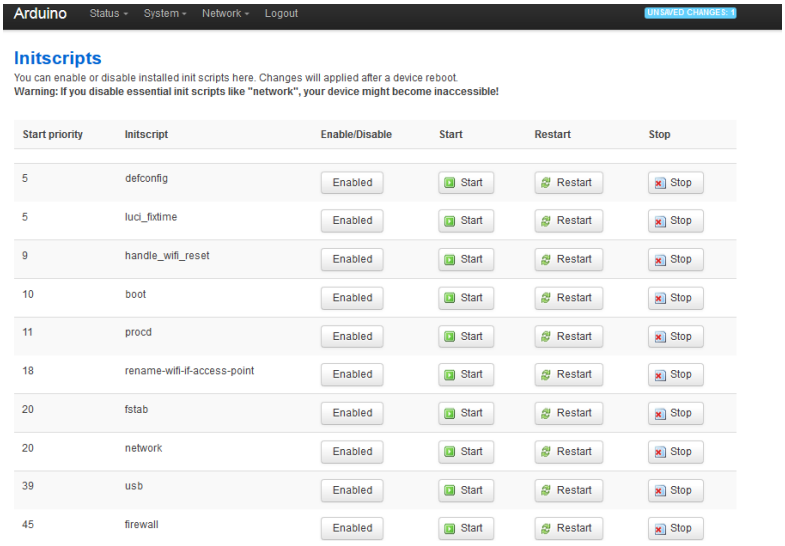


Figura D.2: Menú Startup Luci

Además en la parte inferior existe una opción *Local Startup* (figura D.3) que permite añadir los procesos propios del usuario que deban comenzar al arrancar el Arduino.

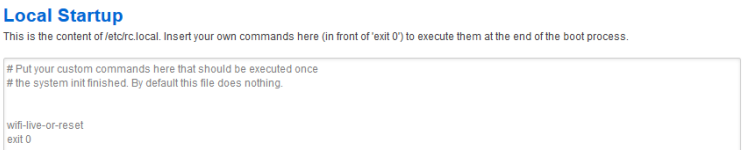


Figura D.3: Local startup

Apéndice E

IpScan23

Al configurar el Arduino para operar como un cliente dentro de la red, en muchas ocasiones no conoceremos su dirección IP. Para ello podemos recurrir a cualquier programa escaneador de redes, en concreto en este proyecto se ha empleado el IpScan23 .

Este programa puede ser descargado de forma gratuita del siguiente enlace:

`http://www.herdprotect.com/ipscan23.exe`

Una vez instalado, al ejecutarlo aparecerá una imagen como la de la figura F.8. Se debe introducir en la barra de superior el rango de direcciones que se desea escanear y hacer click en *Scan*.

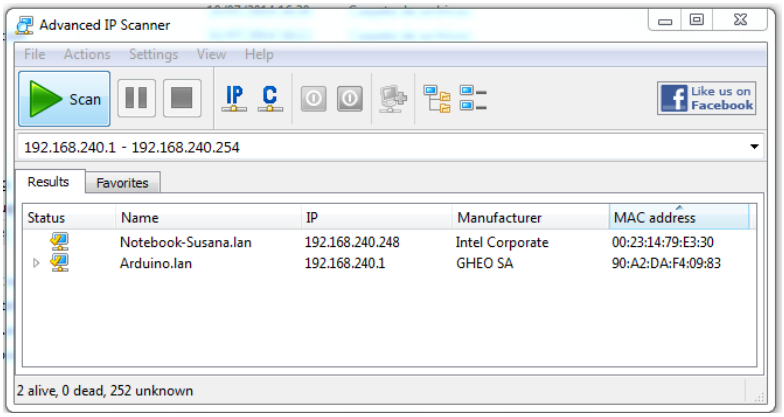


Figura E.1: IpScan23

Tras unos segundos aparecerán todos los dispositivos conectados a esa red, el Arduino será aquel cuyo Manufacturer sea *GHEO SA*.

Nota: En caso de no conocer el rango de direcciones que debe escanearse, se puede abrir una ventana de comando en Windows y recurrir a la sentencia *Ipconfig*

Nota: Es muy importante que el Pc que se esté empleando esté conectado a la misma red que Arduino, ya que de otro modo este proceso no funcionará.

Apéndice F

Planos de las piezas de la carcasa

F.1. Pared lateral

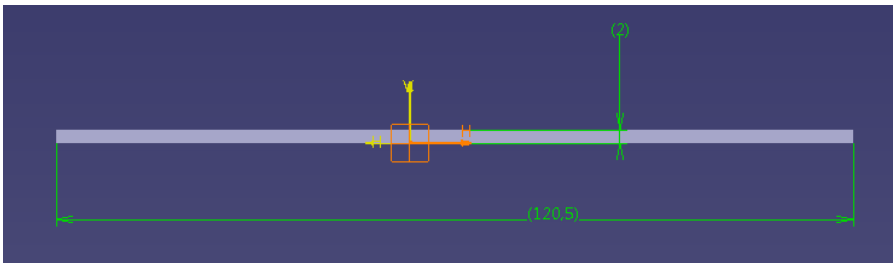


Figura F.1: Planta pared lateral

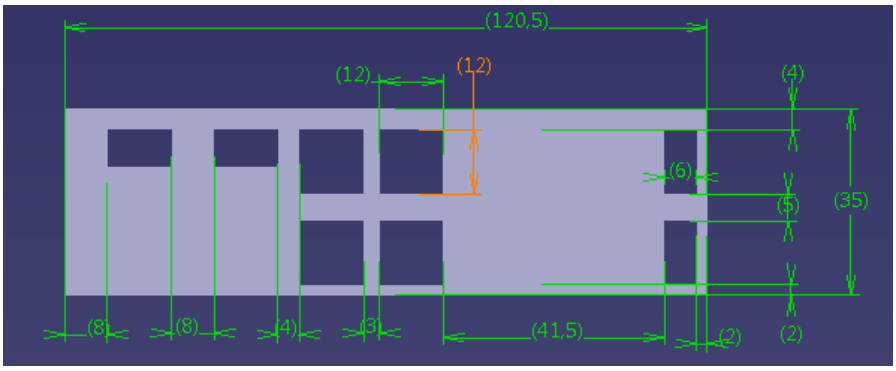


Figura F.2: Alzado pared lateral

F.2. Pared exterior estándar

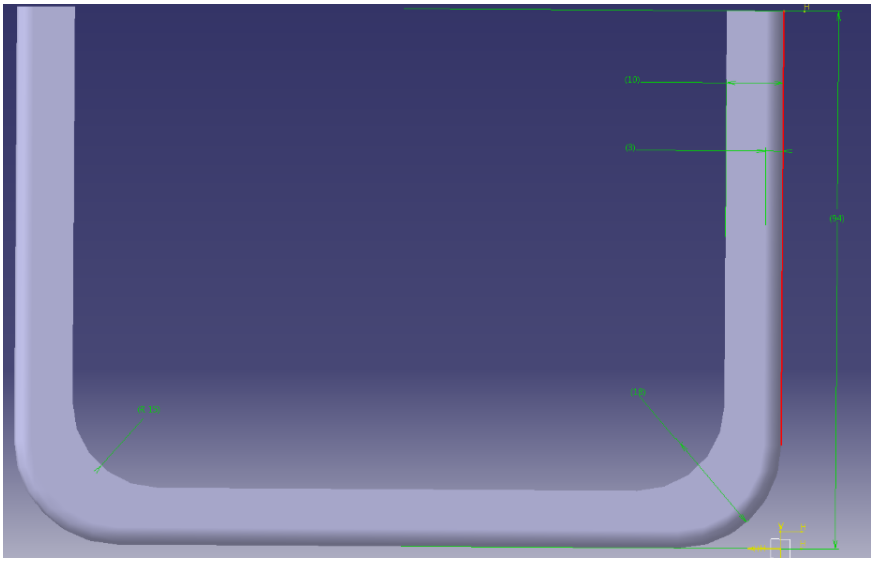


Figura F.3: Planta pared exterior estándar

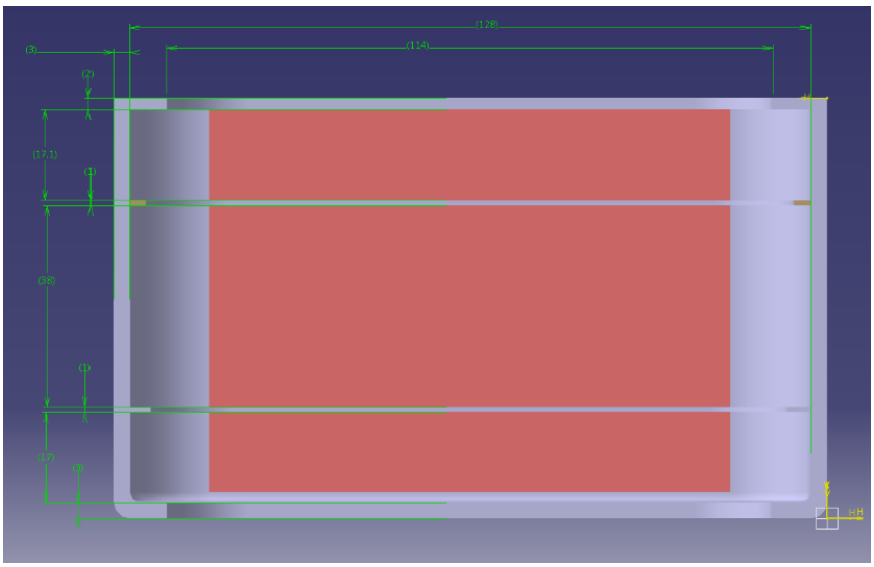


Figura F.4: Alzado pared exterior estándar

F.3. Pared exterior variable

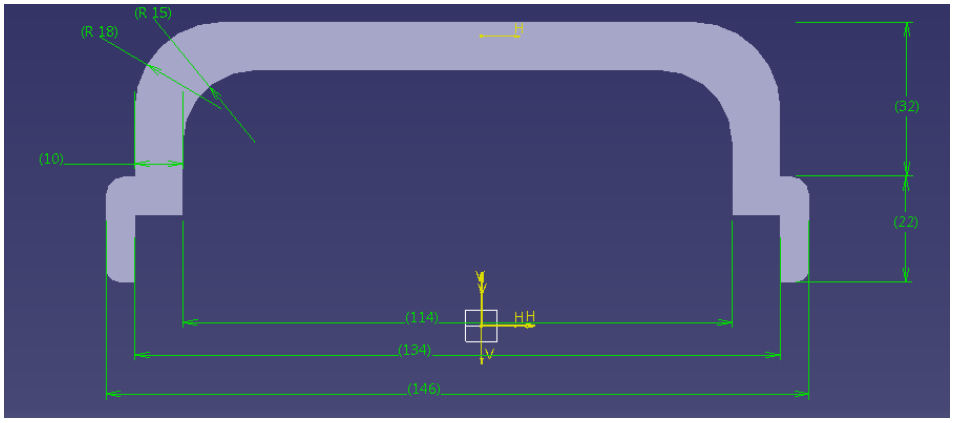


Figura F.5: Planta pared exterior variable

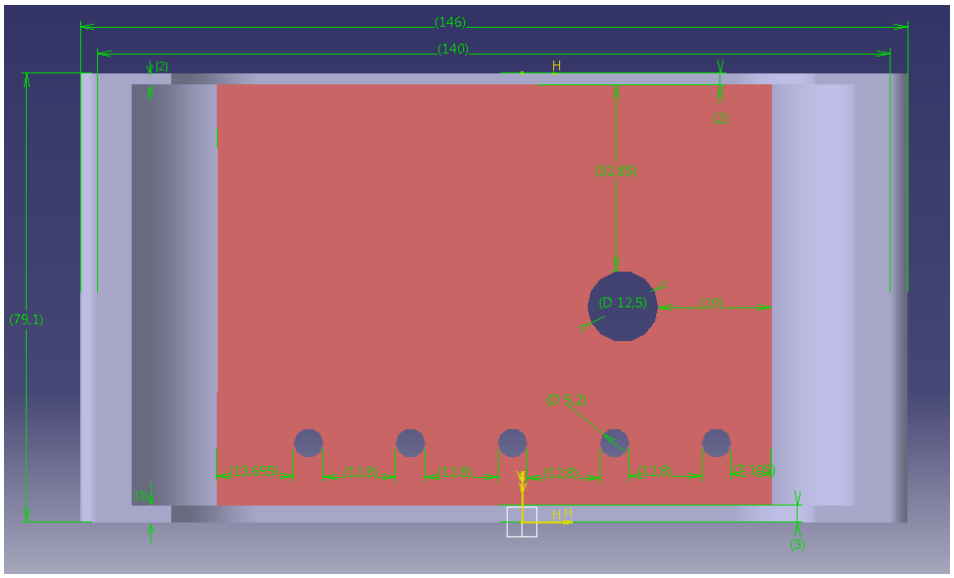


Figura F.6: Alzado pared exterior variable

F.4. Soporte de electrónica

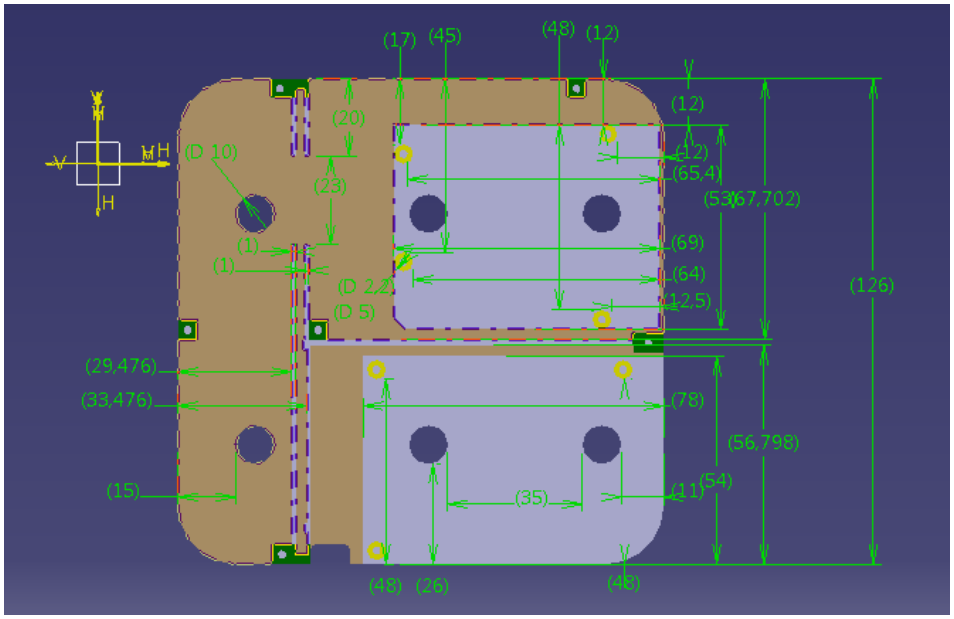


Figura F.7: Planta soporte de electrónica

Figura F.8: Alzado soporte de electrónica

Índice de figuras

Índice de tablas